Special Report 81-1 LEVEL II

12

ADA097770

A PROCESS CONTROL SYSTEM FOR HEWLETT-PACKARD SERIES 21XX COMPUTERS

J. R. Bowman, G. Q. Thorsen, and D. E. Carrell

[redacted]

Approved for public release; distribution unlimited.

81 4 15 002

ERRATA SHEET

Special Report 81-1

A PROCESS CONTROL SYSTEM FOR HEWLETT-PACKARD SERIES 21XX COMPUTERS

J. R. Bowman, G. Q. Thorsen, and D. E. Carrell

The following corrections should be made to subject report.

PAGE NO.	LINE NO.	CORRECTION
12	2	Value vice vlaue
12	9	FO 1 S 5 vice FOL 1 S 5
14	11	values range from 2^0 through 2^{11}... vice values range from 20 through 2^{11}...
18	13	Delete range of -32768 to 32767
21	11	Where X is a state number 0 < X < 31. vice Where X is a state number 0 F X F 31.
23	12	IF 1 R 3 OR AF 5 S vice IF 1 R3 OR AF 5 S.
28	11 (right column)	ST3 AF 10 T ST 1 TH 2 VAR A=A + 1$. vice ST3 AF 10 T ST 1 TH 2 VAR A=A+1$.

Approved for public release; distribution unlimited.

(8)

6) A PROCESS CONTROL SYSTEM FOR HEWLETT-PACKARD SERIES 21XX COMPUTERS

10) John R./Bowman, Gregory Q./Thorsen [redacted] Douglas E./Carrell

14) NAMRL-SR-81-1 9) Special rept.,

16 F51524

Bureau of Medicine and Surgery
17) ZF51 524 004/2011

Naval Air Systems Command
W43-13

Approved by

Ashton Graybiel, M.D.
Assistant for Scientific Programs

Released by

Commander W. M. Houk, MC, USN
Commanding Officer

12) 118

11) Aug [redacted] 80

Naval Aerospace Medical Research Laboratory
Naval Air Station
Pensacola, Florida 32508

406

SUMMARY PAGE

THE PROBLEM

The Vision Research Division of the Naval Aerospace Medical Research Laboratory, needed a computer system to control psychophysical experiments that would remove the need for in-depth knowledge of complex computer languages by investigators. It was required that this system utilize a Hewlett-Packard 2100 series computer.

FINDINGS

The State Diagram System (SDS) was developed to solve this problem. SDS is a tool that can be used by investigators in designing and running psychophysical experiments on Hewlett-Packard's HP-2100 series computers. SDS, as presently designed, is capable of running only those experiments that use discrete inputs and outputs. The system offers the investigator a high level language with which he is already familiar or can easily learn, thus removing the burden of solving these types of problems using more complex computer languages. Written in FORTRAN IV language SDS is an interactive system that does not require assembling or compiling of its programs. The system accepts source language statements from either the system console or disc files and allows the program to be run immediately upon completion of this input process. While SDS does not solve all of the problems encountered in computerizing psychological experiments, its modular design should ease such future modifications as dealing with continuous variables, calling external programs, and controlling multiple experiments.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Ms. Rachel Gadolin, Ms. Karen Venner, and Mr. Delbert Turner, Aerospace Psychology Department, and Mr. Stanley Sulcer, Visual Aids Branch, for their efforts in assembling this report.

The current address for Mr. G. Q. Thorsen is 5315 Bellview Avenue, Pensacola, Florida 32506.
The current address for Mr. D. E. Carrell is 721 N. State St. Apt. 7C, Jackson, Mississippi 39201.

TABLE OF CONTENTS

I. INTRODUCTION

Investigators in the Vision Research Division of the Naval Aerospace Medical Research Laboratory desired to use existing equipment to control on-going and future experiments. This equipment included two Hewlett-Packard computer systems (HP-2100, HP-21MX), each containing a multiprogrammer (HP-6940) and an assortment of input/output equipment. There was no high level language oriented towards process control, and personnel had to learn complex computer languages in order to use the equipment to control experiments. This language barrier was time consuming and often led to delays or minimum use of the computer for experiment control. The following computer system requirements were established by the Vision Research Division to alleviate this problem:

A) Develop a high level language that can be used to program existing computer hardware systems to control psychophysical experiments. This language should be written in terms that investigators are familiar with and should be easily learned by those who may be unskilled in the use of complex computer languages.

B) Implement this system using a high level language such as FORTRAN in order to facilitate changes to other computers in the future.

C) Design the system to control any experiment capable of being controlled by the multiprogrammers' digital input/output cards.

D) The system should be an interactive system that does not require assembling or compiling of the user's program prior to running.

E) The system should be capable of communicating with basic input/output devices such as the console, line printer, paper tape reader, and paper tape punch.

F) Design the system using a modular concept to facilitate future additions of such features as analog input/output or multiple experiment capabilities.

The State Diagram System (SDS) was designed and developed to satisfy the above requirements. SDS is an event driven, table oriented, real-time system that operates under Hewlett-Packard's real-time executive RTE-II. SDS is accurate to within one tick of the RTE-II clock which runs at a frequency of 100 Hz. The purpose of this report is to define the capabilities and limitations of this system.

II. OVERVIEW OF HARDWARE

SDS was designed to be used on the Hewlett-Packard HP-2100 or HP-21MX computer system which includes the following peripheral equipment and logical unit assignments:

A) Console - logical unit 1.

B) Disc - logical unit 2 and 10.

C) Mag Tape - logical unit 8.

D) Multiprogrammer - logical unit 9.

E) Line Printer - logical unit 6.

F) Paper Tape Reader - logical unit 5.

G) Paper Tape Punch - logical unit 4.

The console, disc, magnetic tape, and multiprogrammer are required for the SDS. The paper tape reader, paper tape punch, and line printer are optional equipment; however, the omission of any equipment or changing of logical unit assignments may require minor modifications in the software.

The multiprogrammer, HP-6940B, is an input/output (I/O) control unit that converts a single computer I/O channel into 15 I/O channels if all of its capabilities are utilized. In the existing SDS system, only two channels, an event sense card and a relay output card, are utilized. The event sense card monitors 12 data input lines and notifies the computer when a change occurs in the level of these lines. The relay output card contains 12 output relays that can be energized or de-energized by the computer. In addition to the multiprogrammer, HP-6940B, it is possible to install up to 15 extender units, HP-6941B, which would have the capability of converting a single computer I/O channel into 240 I/O channels. It should again be noted that any change in the existing multiprogrammer capabilities would require modifying the software.

The multiprogrammer is capable of housing the following types of I/O cards in either the main unit or the extender units:

A) Event sense.

B) Digital input for counter with interrupt.

C) Digital I/O.

D) Digital input only.

E) Analog output.

F) Timers.

G) Pulse counters.

The number of each of these cards is optional and, as can be seen by the types of cards available, the need of costly special interface devices to control an experiment could very often be eliminated. Refer to Hewlett-Packards HP-6940B Operating and Service Manual for detailed descriptions of the capabilities of each of these cards.

III. OVERVIEW OF SOFTWARE

A. Hewlett-Packard Real-Time Executive (RTE-II)

Multiprogramming using the RTE-II system requires that programs be installed in the system during system generation if more than two programs are required in core at any given time. Since SDS was designed in a modular manner, it was desirable to develop a method by which these modules or tasks could be loaded into core simultaneously, using RTE-II's loader. This capability would eliminate the requirement of a new system generation each time SDS was modified. To accomplish this, eight dummy programs were installed during system generation. These programs were named T1XXX through T8XXX, indicating the task numbers and complying with the ISA FORTRAN Extension Package requirement that the last three characters must be X's. This requirement only exists in RTE-II when the event Sense Interface routine is being used to schedule tasks. These dummy programs can be any simple programs, as shown in Appendix A, and serve only to establish ID Segment maps in the system. Word eight of these ID Segment maps contain the primary entry points of programs T1XXX through T8XXX and is the only word that needs to be altered before scheduling the tasks. Subroutines NTASK and NTSK1 through NTSK8 modify word eight of these ID Segment maps at run time, thus allowing up to ten programs to be loaded into core by RTE-II's loader. Refer to Appendix A for a detailed description of this process. With this exception, the RTE-II system is intact and is described in the RTE-II operating manual.

B. State Diagram System

SDS was developed to provide automatic control of psychophysical experiments, using discrete inputs and outputs. Similar to SKED and ACT-INTERACT systems available for DEC and NOVA computers, a high level language familiar to investigators is used, thus eliminating their need for in-depth knowledge of complex computer languages. The fundamental idea behind SDS is that experiments using discrete inputs and outputs can be broken down into basic elements that modify the subject's environment. These basic elements usually deal with the presentation of stimuli, the detection of responses, and the measurement of elapsed time, which makes them easily automated using the computer. In addition to automating experiments SDS logs each event on magnetic tape, thus allowing off-line analysis of the experiment's data at a later date. SDS was designed in a modular form, as shown in Figure 1, to facilitate future modifications such as multiple experiment control or the ability to control experiments using continuous variables. A brief description of each of the programs used in SDS follows. Listings of the programs and subroutines of SDS are included in Appendix B.

Program OPCOM is the operator communications program. OPCOM accepts the SDS source language statements from either the disc or the system console. OPCOM decodes each line of source language statements and calls upon program DOOPS to load each decoded instruction into the proper tables of the experiment controller program EXPR. When the end of the source language statements is reached, the complete program has been decoded and loaded. OPCOM then schedules program SDS to start the experiment.

Program DOCPS is used to communicate the decoded SDS instructions between programs OPCOM and EXPR. This program crosses the foreground boundary into background and sets up the proper tables with each decoded instruction supplied by OPCOM.

Program RDWRT performs read and write functions to disc files OPIN and OPOUT, respectively. If instructed to read source language statements from the disc, RDWRT opens disc file OPIN, reads single lines of source language statements, and passes each line to program OPCOM for decoding. This process is performed until the end of disc file OPIN is reached. RDWRT also writes each line of source language statements in disc file OPOUT. This write function occurs when the system console or when disc file OPIN is used for inputting source language statements. The results of this write function is that file OPOUT always contains a copy of the most recent SDS program. File OPOUT can be saved for future use or can be transferred to disc file OPIN for running the same SDS program using the disc as an input device. In addition to creating disc file OPIN in this manner RTE-II's editor can be used to create or modify source language programs named OPIN.

Program SDS performs the functions of initialization, starting experiments, and ending experiments. Prior to OPCOM accepting source language statements from either disc or the system console program SDS is called upon to initialize all variables and tables within the system. When notified by OPCOM to start an experiment program, SDS issues the start of experiment event code. When notified by program EXPR that the experiment has ended program, SDS terminates all active programs, including itself, and returns control to RTE-II's file manager program FMGR.

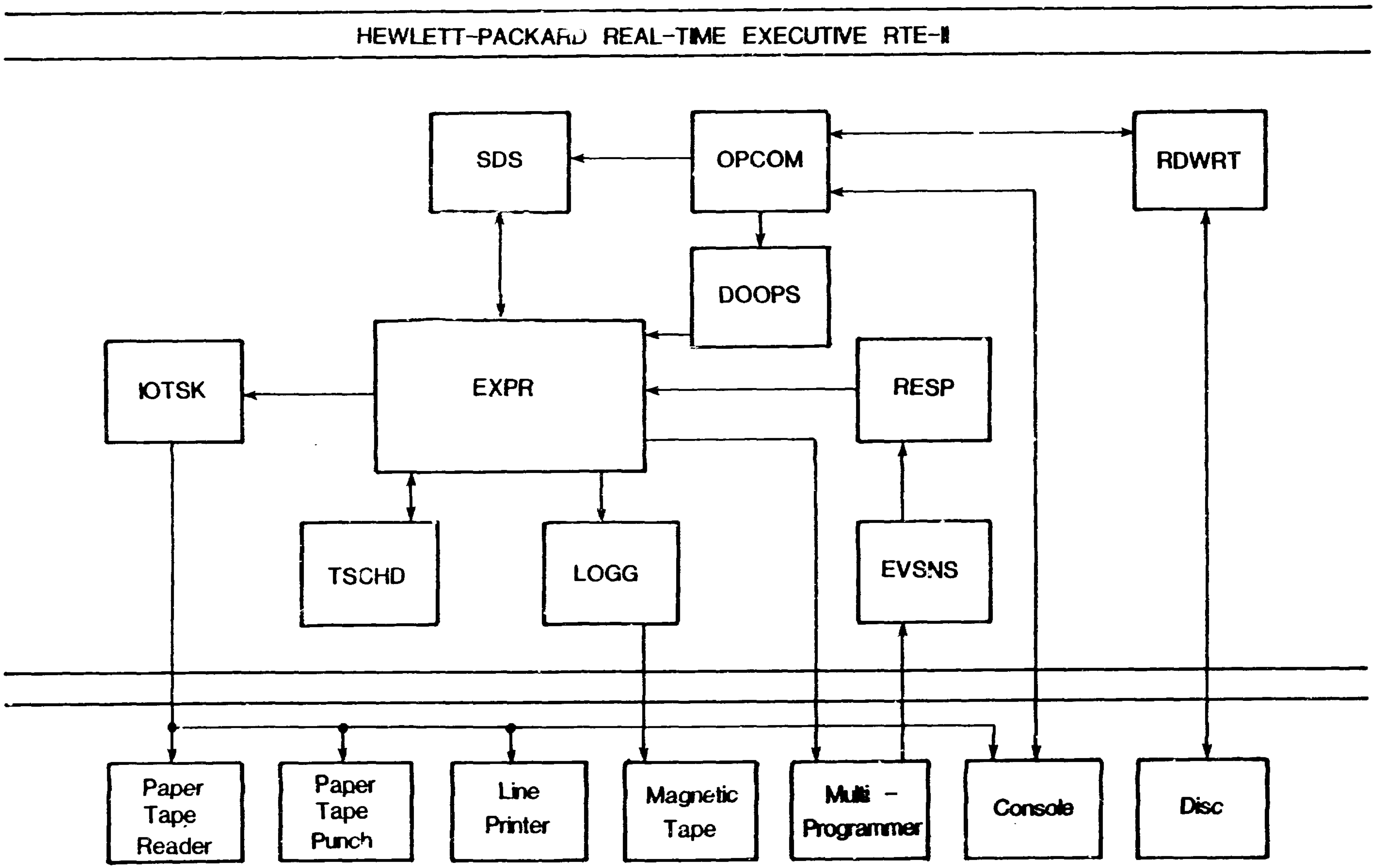


Figure 1. Block Diagram SDS

When a switch closure occurs on any one of the 12 event sense data input lines, the event sense interface routine, EVSNS, schedules program RESP. Program RESP then determines which input line caused the interrupt, issues a response event code, and passes the switch number to program EXPR.

Program TSCHD maintains a time event schedule. When a requested time has elapsed, program TSCHD issues a time event code and passes the necessary information needed to service the time event to program EXPR.

Program IOTSK performs I/O operations requested by the experiment controller task EXPR. The I/O operations that can be performed by SDS are input from the paper tape reader and output to the system console, line printer, and paper tape punch.

Each event that occurs during the running of an SDS program is logged on the magnetic tape with sufficient information to identify the event. This function is performed by program LOGG on a low priority basis.

Program EXPR is the overall experiment controller program. EXPR is driven by the start of experiment, response, time, and relational events. This program takes action on these events as directed by the SDS Program. Upon completion of the SDS program, EXPR notifies program SDS to terminate all active programs.

IV. NOTATION SYSTEM

Prior to describing the instruction set of SDS it is important that the notation system be introduced. This system of notation should be studied carefully because proper diagramming of the experiment and using proper notation result in the programming function being accomplished automatically.

The state is the basic unit of the notational system and is used to represent one element of a discrete input/output experiment. The state diagram is drawn in Figure 2. The state number is drawn into the box in the upper right corner of the state diagram as represented by the X. The Ys represent one or more of the SDS input/output instructions, substate instruction, then instruction, initialize variable instruction, dimension statement, or the stimulus instruction. The Z represents any one of the four basic instructions of the SDS that will cause a transition from the state, and the → represents the direction of program flow or transition. The four instructions that will cause a transition in the SDS are the AFTER, FOLLOWING, IF, and modified IF instructions. A transition is the exiting from one state and the entry into the designated next state and is considered to be an instantaneous event.

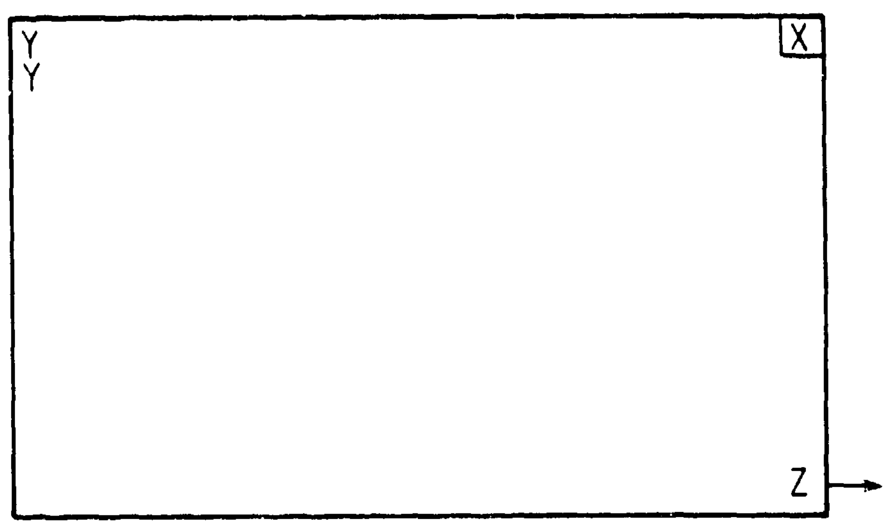


Figure 2. State Notation System

One state can be a substate of another state. The substate notation is to draw the state diagram nested within another state as shown in Figure 3 In this example state 2 is a substate of state 1 as denoted by the use of the same left border line and by the fact that state 2 is drawn within the boundary of state 1. It should also be noted that the SUBSTATE instruction SS 2 has been entered in the upper left corner of state 1. State 3 is a substate of state 2 for the same reasons and is included in this drawing to demonstrate the capabilities of multiple nestings.

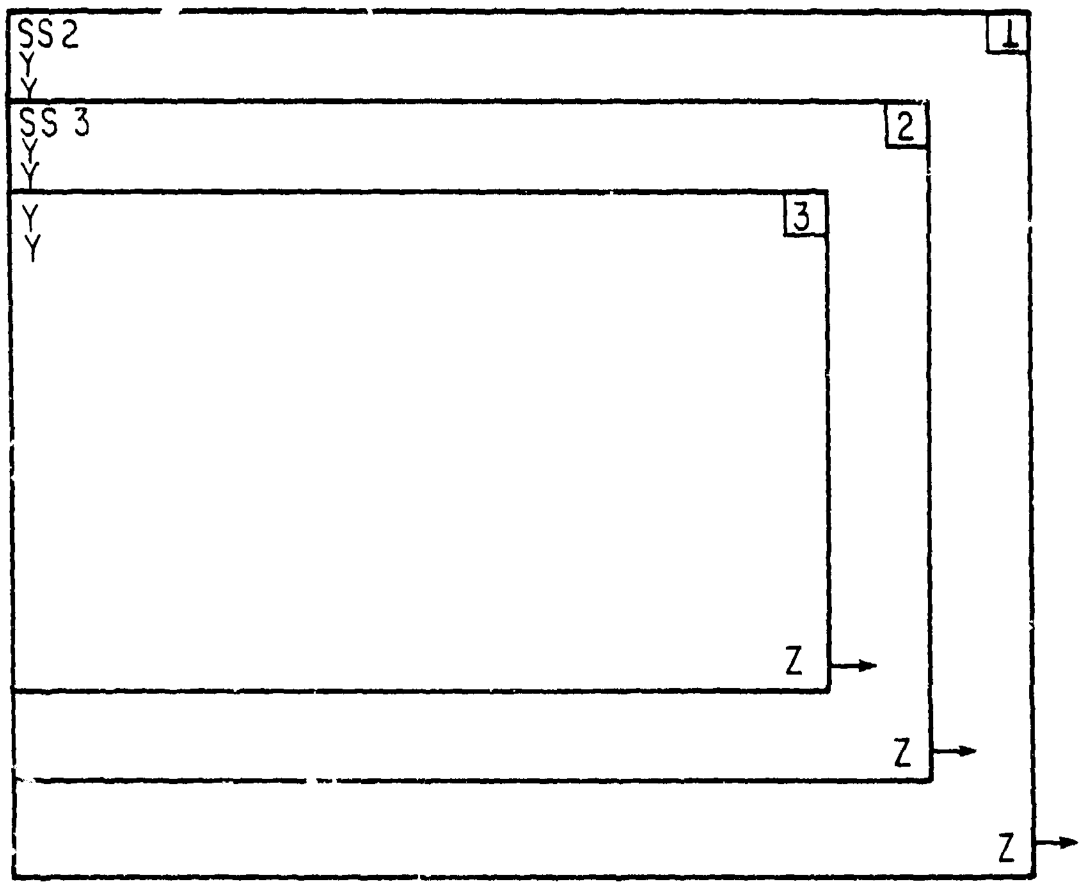


Figure 3. Substate Notation

The direction of program flow is represented by the (right arrow). When states are connected by an → as shown in Figure 4, the THEN statement should be entered in the upper left corner of the state.

Both the SUBSTATE instruction and the THEN instruction are implied by the manner in which they are drawn. For example, if a state is drawn within another and uses its left border, the SUBSTATE instruction is implied, and if two states are connected by an arrow, the THEN instruction is implied. It is not necessary to write these instructions in the upper left corner of each state; however, caution should be taken when translating the state diagram into the actual SDS program.

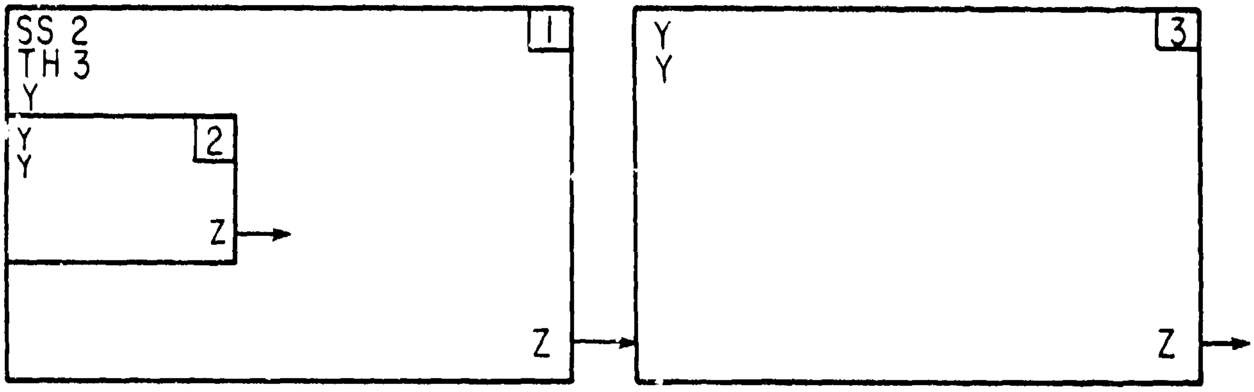


Figure 4. Connecting State Notation

This concludes the basic SDS notation system. It is recommended that this section be reviewed after reading Section V describing the SDS instruction set. The success in using the SDS lies in the care taken in diagramming the experiment. Since each instruction used within a state is written into the diagram, the programming function is merely to transform the diagram into proper SDS source language format.

V. THE SDS INSTRUCTION SET

There are five categories of intructions in the SDS instruction set. These are job control, transitional, modifying or identifying, logical, and input/output instructions. Job control instructions determine the beginning and the end of the program and the end of each line of source language statements. Transitional instructions define the event that will cause a transition from one state to the next state. Modifying or identifying instructions modify or identify variables, states, substates, stimuli, or direction of program flow. Logical instructions supply the capability of tying two or more transitional instructions together in a single state though the use of a logical OR or a logical AND. Input/output instructions are used to control input/output to or from the system console, paper tape reader, paper tape punch, and the line printer.

A. <u>Job Control</u> - there are three job control statements in the SDS language. These statements are $, NEW, and END. Each of these instructions is used in every source language program.

1. $ - The $ is used to terminate each source language line in an SDS program. The operator communication program uses the $ to delineate the end of line. If the $ is omitted, the operator communications program will combine two or more lines of code, resulting in program errors.

2. NEW - The NEW instruction is required by the SDS to initiate a dialog concerning the source of input of source language code. The instruction must be the first statement in every source language program and is written as:

NEW$

3. END - The END instruction is required by the SDS to initiate a dialog concerning running of the program. The instruction must be the last statement in every source language program and is written as:

END$

B. Transitional - There are four transitional instructions in the SDS language. These instructions are AFTER, FOLLWING, IF, and a modified IF or relational instruction. These instructions determine when the SDS program will exit one state and make a transition to the next state in the program.

1. AFTER - The AFTER instruction leaves the state it is in and makes a transition to the next state in the program after passage of a designated amount of time. The AFTER instruction is written as:

$$AF_{\Lambda}X_{\Lambda}Y$$

Where X is a constant, a variable, or an element of an array; the value of X must be greater than Ø and must not exceed 32767; and Y is one of the designators T, S, M, or H that designate ticks (10s of ms), seconds, minutes, or hours, respectively. If X is a variable or an array element, it is considered to be X number of seconds and must have been previously defined in the SDS program. NOTE: When writing source language programs using the SDS, it is necessary to comply with certain syntax restrictions to inform the operator communications program of the beginning and end of source statements. One such restriction is the use of the delineator blank which will be designated by the symbol Λ in all of the following examples. When creating source language programs, whether by using the editor or by input from the console keyboard, this delineator must be used as shown in each instruction description. It should also be noted that every instruction within a source language program must be separated by a blank. The SDS operator communication package is searching the source language line to find characters that indicate that an instruction has been reached, such as the two characters AF in the AFTER instruction. It then skips all characters until a blank is found which allows the instruction AFTER to be written in its entirety if so desired. With the exception of input/output instructions only the first two characters of each instruction need be typed. When using input/output instructions, the three designated characters must be typed.

Examples of AFTER instruction:

AF 5 T - exit this state after 5 ticks of the clock (50 ms)

AF 8 S - exit this state after 8 seconds

AF 3 M - exit this state after 3 minutes

AF 2 H - exit this state after 2 hours

AF A S - exit this state after A seconds

AFTER 1 T - exit this state after 1 tick of the clock

AFXYZLMN 1 T - exit this state after 1 tick of the clock

NOTE: SDS ignores misspelled words if the first two characters are correct and there are no imbedded blanks.

The AFTER instruction is diagrammed in Figure 5 in which state #1 will exit after 3 minutes.

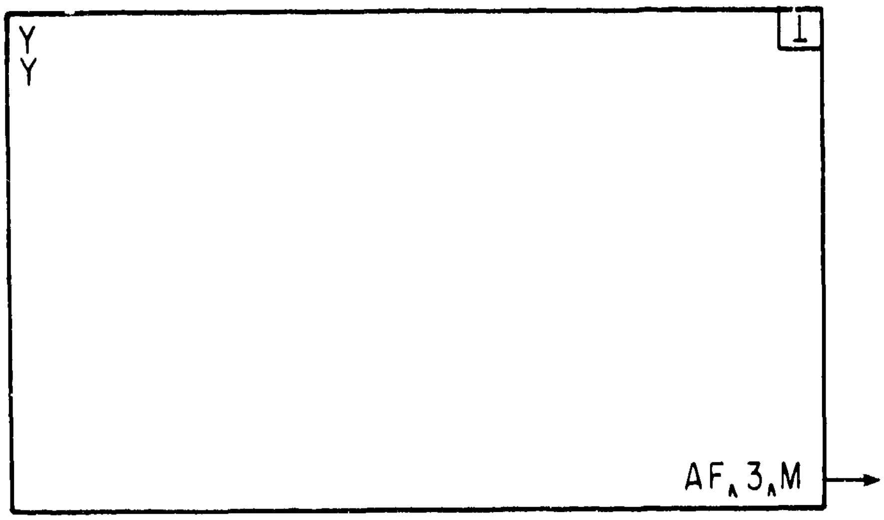


Figure 5. Diagramming the AFTER Instruction

2. FOLLOWING - the FOLLOWING instruction leaves the state it is in and makes a transition to the next state in the program upon occurrence of a designated count X of a designated state Y that must be a state other than itself. The FOLLOWING instruction is written as:

FOL X S Y

Where X is a constant, a variable, or an element of an array designating the desired count; the vlaue of X must be greater than ∅ and must not exceed 32767, S is the character S; and Y is a constant, a variable, or an element of an array designating the state number of the state that is going to be counted. If X or Y is a variable or an array element, it must have been previously defined in the SDS program. The state number designated by Y must be a valid state number other than its own.

Examples of FOLLOWING instruction:

FOL 1 S 5	- exit this state following the occurrence of 1 state #5
FOL A S 3	- exit this state following the occurrence of A state #3
FOL 3 S A	- exit this state following the occurrence of 3 state #A
FOL A S B	- exit this state following the occurrence of A state #B
FOL 1 S 5	- exit this state following the occurrence of 1 state #5
FOLLOWING 1 S 5	- exit this state following the occurrence of 1 state #5

The FOLLOWING instruction is diagrammed in Figure 6 in which state #1 will exit following the occurrence of five state #3. Note that states #2 and #3 oscillate between each other every second.

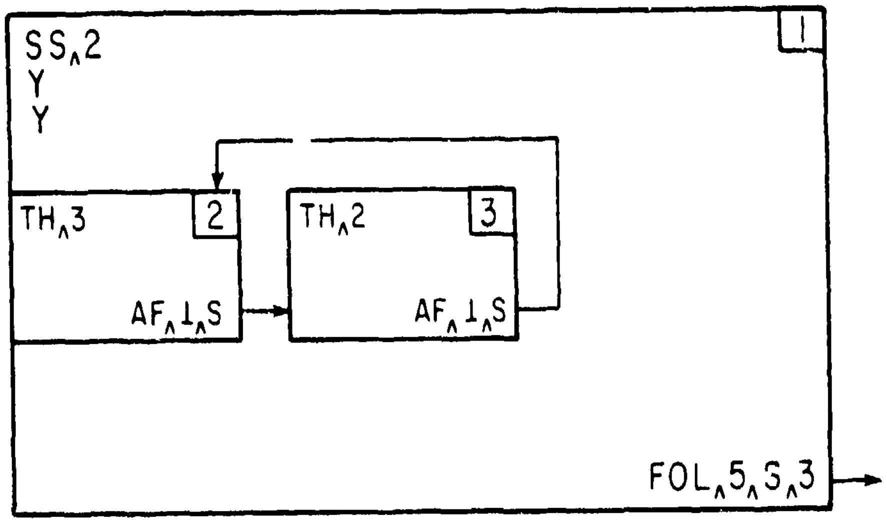


Figure 6. Diagramming the FOLLOWING Instruction

3. IF - The basic IF instruction leaves the state it is in and makes a transition to the next state in the program upon occurrence of a designated number X of correct responses whose number Y is also designated in the instruction. The basic IF instruction is written as:

IF X R Y

Where X is a constant, a variable, or an element of an array designating the desired count; the value of X must be greater than Ø and must not exceed 32767; R is the character R; and Y is a constant, a variable, or an element of an array designating the response number desired. If X is a variable or an array element, it must have been previously defined in the SDS program. If Y is a variable or an array element, it must have been a valid response number previously defined in the SDS program.

NOTE: Response numbers range from respone #1 through response #12 which have a direct relationship to the bit position of the response word. For example, response #4 implies the fourth bit of the response word.

Examples of basic IF instruction:

IF 3R 5	- exit this state if 3 response #5 occurs
IF A R B	- exit this state if A response #B occurs
IF A(1) R B(4)	- exit this state if A(1) response #B(4) occurs
IF A(D) R B(E)	- exit this state if A(D) response #B(E) occurs

The basic IF instruction is diagrammed in Figure 7 in which state #1 will exit if response #4 occurs 5 times.

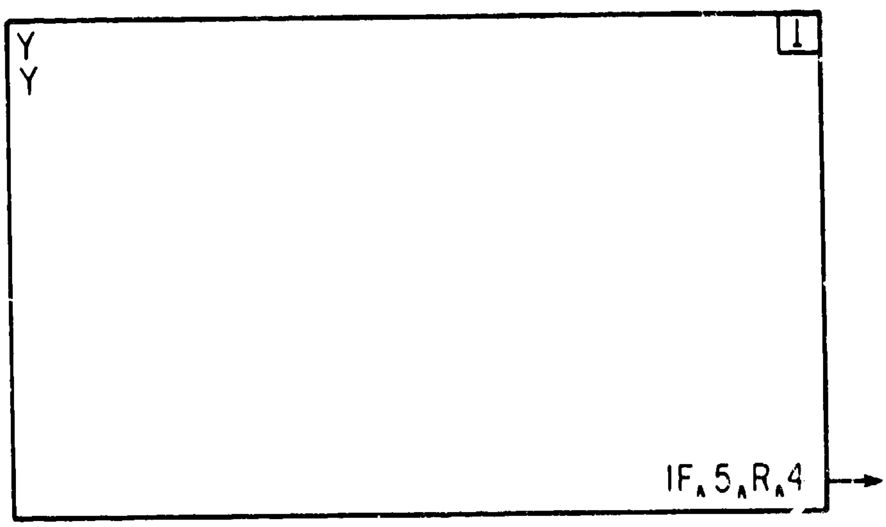


Figure 7. Diagramming the Basic IF Instruction

Another form of the IF instruction is the binary IF instruction which is designed to detect multiple responses. The binary IF instruction leaves the state it is in and makes a transition to the next state in the program upon occurrence of a designated number of correct response patterns whose bits are also designated in the instruction. In the basic IF instruction, as previously stated, the response number implies the bit position of the response word as shown in Figure 7; however, if the response number 3 was designated in the binary IF instruction, it would be made of bits 0 and 1 whose values are 2 and 2^1, respectively. The decimal value must be used to designate the proper value for the desired multiple response. In the response word there are 12 bits whose values range from 2o through 2^{11} as shown in Table I. The binary IF instruction is written as:

IF X RB Y

Where X is a constant, a variable, or an element of an array designating the desired count; the value of X must be greater than ∅ and must not exceed 32767; RB are the characters RB; and Y is a constant, a variable, or an element of an array designating the response pattern desired; the value of Y must be greater than ∅ and must not exceed 4095. If X or Y is a variable or an array element, it must have been previously defined in the SDS program.

Table I

BIT Values Used in Multiple Response Word

BIT #	VALUE	RESPONSE #
0	1	1
1	2	2
2	4	3
3	8	4
4	18	5
5	32	6
6	64	7
7	128	8
8	256	9
9	512	10
10	1024	11
11	2048	12

Examples of binary IF instruction:

IF 1 RF 3 - exit this state if 1 response pattern 3 occurs

IF A RB 3 - exit this state if A response pattern 3 occurs

IF A(1) RB 3 - exit this state if A(1) response pattern 3 occurs

IF A(B) RB 3 - exit this state if A(B) response pattern 3 occurs

IF A RB B - exit this state if A response pattern B occurs

The binary IF instruction is diagrammed in Figure 8 in which state #1 will exit if response pattern 2049, bit Ø and bit 11, occurs five times.

4. IF (modified) - The modified IF instruction or relational instruction leaves the state it is in and makes a transition to the next state in the program upon the satisfaction of the relationship of its two variables. The optional relational operators are EQ, NE, LT, GT, LE, and GE, and the respective mathematic functions are equal to, not equal to, less than, greater than, less than or equal to, and greater than or equal to. The relational instruction is written as:

IF X Z Y

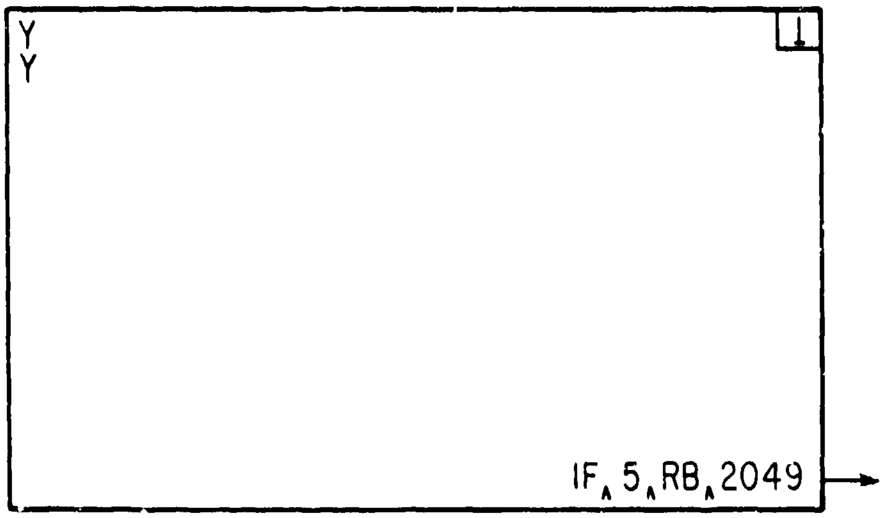


Figure 8. Diagramming the Binary IF Instruction

Where X is a variable or an array element that has been previously defined in the SDS program; the value of X must be in the range of -32768 to 32767; Z is one of relational operators EQ, NE, LT, GT, LE, or GE; and Y is a variable or an array element that has been previously defined in the SDS program; the value of Y must be in the range of -32768 to 32767.

Examples of relational instruction:

IF A EQ C - exit this state if A is equal to C

IF D GE Y - exit this state if D is greater than or equal to Y

IF A LT B - exit this state if A is less than B

The relational instruction is diagrammed in Figure 9 in which state #1 will exit if variable A is equal to variable B.

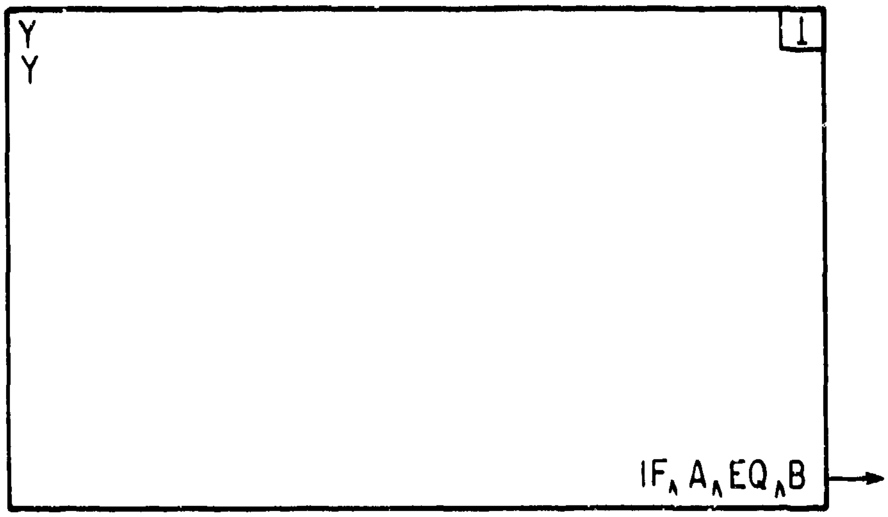


Figure 9. Diagramming the Relational Instruction

C. <u>Modifying or Identifying</u> - There are six instructions that can modify or identify direction, variables, array size and content, stimuli, and states relationship to other states; i.e., state or substate. These instructions are THEN, VARIABLE, DIMENSION, STIMULUS, STATE, and SUBSTATE, respectively.

1. THEN - The THEN instruction determines the direction the program will follow in the SDS program and is written as:

TH X

Where X can be a constant, a variable, or an element of an array designating the next state to be entered when an exit is made from this state. If X is a variable or an array element, it must have been previously defined in the SDS program. The state number designated by X must be a valid state number.

Examples of THEN instruction:

TH 2	- when this state exits, go to state 2
TH A	- when this state exists, go to state A
TH A(1)	- when this state exits, go to state A(1)

TH A(B) - when this state exits, go to state A(B)

THEN 2 - when this state exists, go to state 2

The THEN instruction is diagrammed in Figures 10(a) and 10(b). In Figure 10(a) the THEN instruction is written in the upper left corner of the state diagram, and in Figure 10(b) the THEN statement is implied by the arrow. Both Figures 10(a) and 10(b) perform the same function, which is to go to state #2 when state #1 exits after one second.

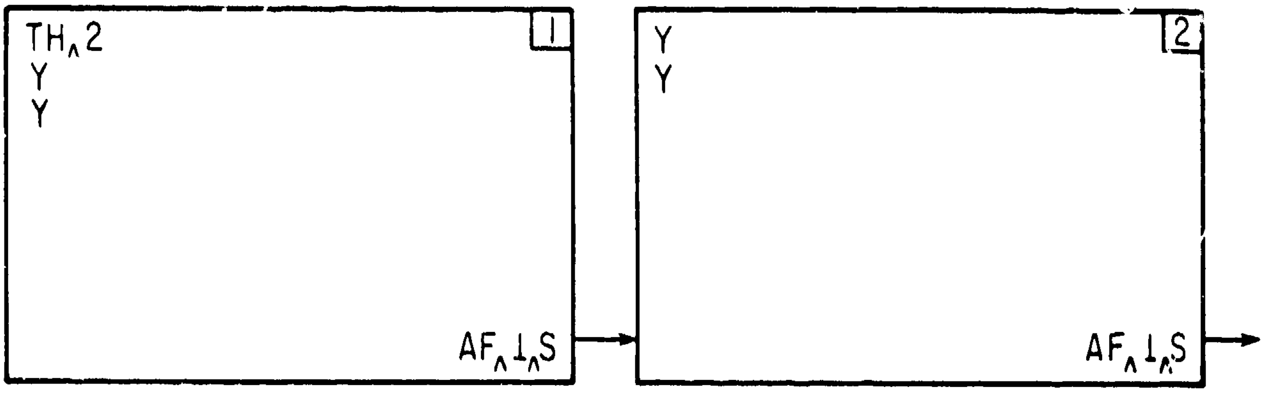


Figure 10(a). Diagramming the THEN Instruction Using Written Notation

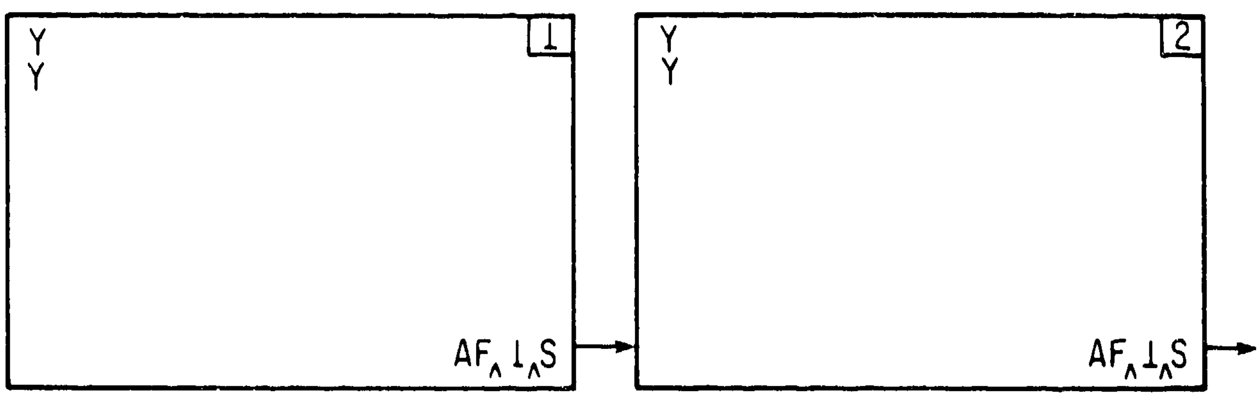


Figure 10(b). Diagramming the THEN Instruction Using Implied Notation

2. VARIABLE - The VARIABLE instruction is used to define one or more variables to be used in the SDS program. The instruction must be the last instruction in a line of source statements used to describe a state. There are 25 variables available to the user, and they must be designated as A through Y. Variable Z is presently being used by the SDS. The VARIABLE instruction is written as:

VAR X = Y or VAR X = Y, X = Y, etc.

Where X is one of the letters A through Y used to designate the desired variables and Y is a constant, a variable, or an element of an array defining the value of the designated variable. If Y is a variable or an array element, it must have been previously defined in the SDS program. The value of Y must be in the range of -32768 to 32767.
range of -32768 to 32767
Examples of VARIABLE instruction:

VAR A = 1 - variable A is set equal to 1

VAR A = B - variable A is set equal to B

VAR A = B(1) - variable A is set equal to B(1)

VAR A = B(C) - variable A is set equal to B(C).

VAR A(1) = 1 - array element A(1) is set equal to 1

VAR A(B) = 1 - array element A(B) is set equal to 1

The VARIABLE instruction is diagrammed in Figure 11 in which variables A and B are both set equal to five.

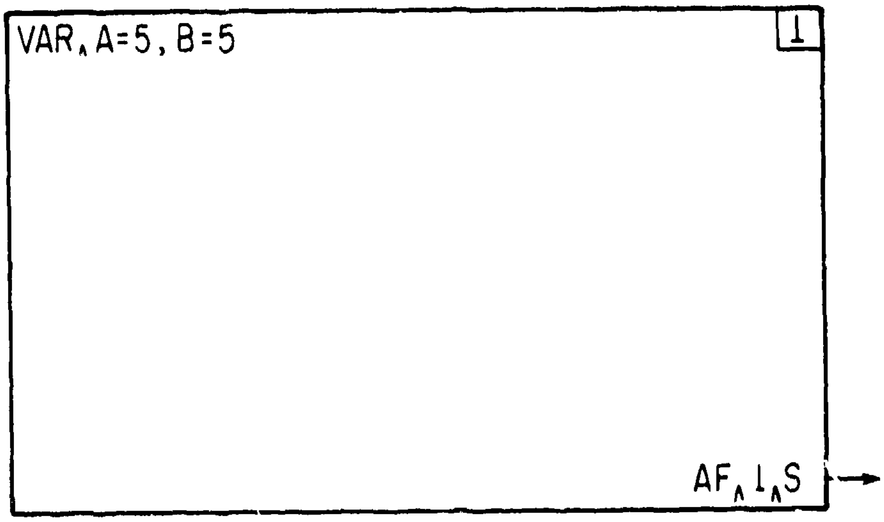


Figure 11. Diagramming the VARIABLE Instruction

3. DIMENSION - The DIMENSION instruction is used to define up to four arrays. The combined size of the four arrays must not exceed 200 words. The array defined must be named by any one of 26 available names which are the characters A through Z. All arrays must be defined using a DIMENSION instruction prior to any attempt to address an element of the array. The DIMENSION instruction is written as:

DIM X,Y

Where X is the name of the array and must be any one of the 26 characters A through Z; and Y is the number of words in the array. It should be again noted that if more than one array is defined, the combined total of the size Y of all of the arrays must not exceed 200 words.

Examples of DIMENSION instruction:

DIM A,50 - array named A is defined as being 50 words long

DIM B, 150 - array named B is defined as being 150 words long

NOTE: The combined size of arrays A and B does not exceed 200 words in length.

The DIMENSION instruction is diagrammed as shown in Figure 12 in which two arrays are named A and B and are defined as being 50 and 150 words in length, respectively.

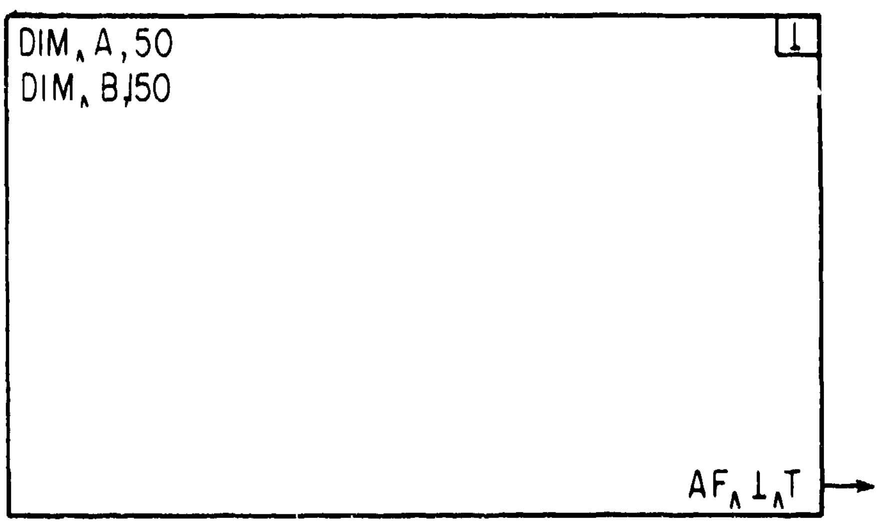


Figure 12. Diagramming the DIMENSION Instruction

4. STIMULUS - The STIMULUS instruction is used to issue a desired stimulus during a state. The stimulus is turned on when a state is entered and is turned off when a state ends. The value of the stimulus data word determines which and how many stimuli will be issued. The decimal value must be used to designate the proper value for the desired stimuli. There are 12 bits in the stimulus word and their values are as shown in Table II.

Table II

BIT Values Used in Stimulus Word

BIT #	VALUE	STIMULUS
0	1	1
1	2	2
2	4	3
3	8	4
4	16	5
5	32	6
6	64	7
7	128	8
8	256	9
9	512	10
10	1024	11
11	2048	12

The STIMULUS instruction is written as:

ST X

Where X is a constant, a variable, or an element of an array designating the desired stimuli. If X is a variable or an array element, it must have been previously defined in the SDS program. The value of X must be greater than Ø and must not exceed 4095.

Examples of STIMULUS instruction:

ST 1 - issue stimulus bit 1 during this state

ST 3 - issue stimuli bits 1 and 2 during this state

ST A - issue stimuli bits in variable A during this state

ST A(1) - issue stimuli bits in array element A(1) during this state

ST A(B) - issue stimuli bits in array element A(B) during this state

NOTE: If the characters ST are confusing because of the STATE instruction, the characters SB can be used to replace ST in the STIMULUS instruction.

The STIMULUS instruction is diagrammed as shown in Figure 13 in which stimulus bit #1 is turned on during the entire time state #1 is active.

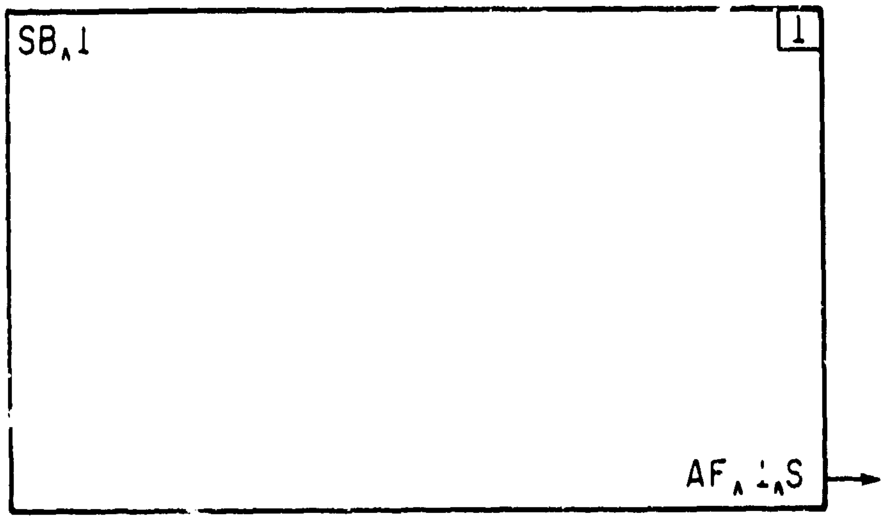


Figure 13. Diagramming the STIMULUS Instruction

5. STATE - The STATE instruction is used to assign a state number to each state. It must be the first instruction in every source line, and X must be a constant greater than Ø and less than 31. The STATE instruction is written as:

ST X

STATE X

Where X is a state number 0 F X F 31.

6. SUBSTATE - The SUBSTATE instruction is used to declare another state to be a substate of the state in which the SUBSTATE instruction appears. The SUBSTATE Instruction is written as:

SU X

Where X is a constant and must be a valid state number of an existing state in the program in which the SUBSTATE instruction appears. The character SS can be used to replace the characters SU if it is desired. The value of X must be greater than Ø and less than 31.

Examples of SUBSTATE instruction:

SU 4 - state 4 is a substate of this state

SU 3 - state 3 is a substate of this state

SS 3 - state 3 is a substate of this state

The SUBSTATE instruction is diagrammed in Figure 14(a) in which state #2 is declared to be a substate of state #1. Figure 14(b) makes the same declaration except it implies that state #2 is a substate of state #1 by the fact that they both use the same left border and state #2 is nested within state #1.

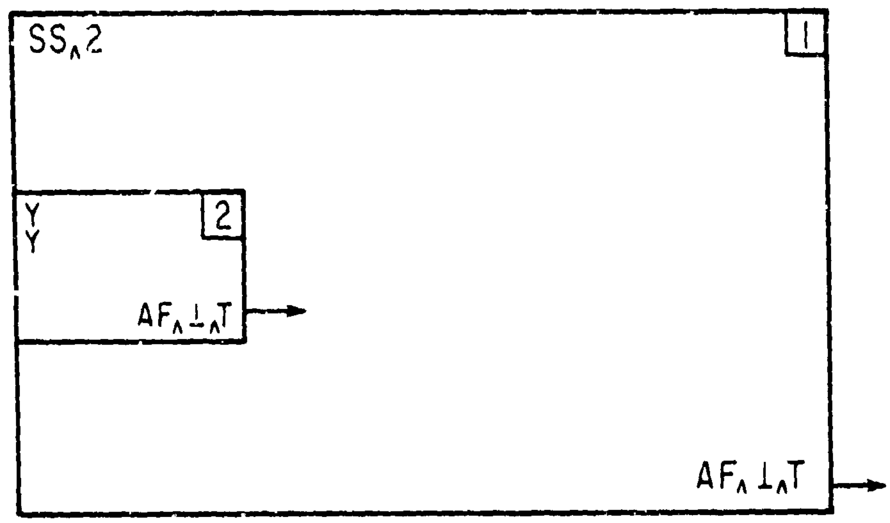


Figure 14(a). Diagramming the SUBSTATE Instruction Using Written Notation

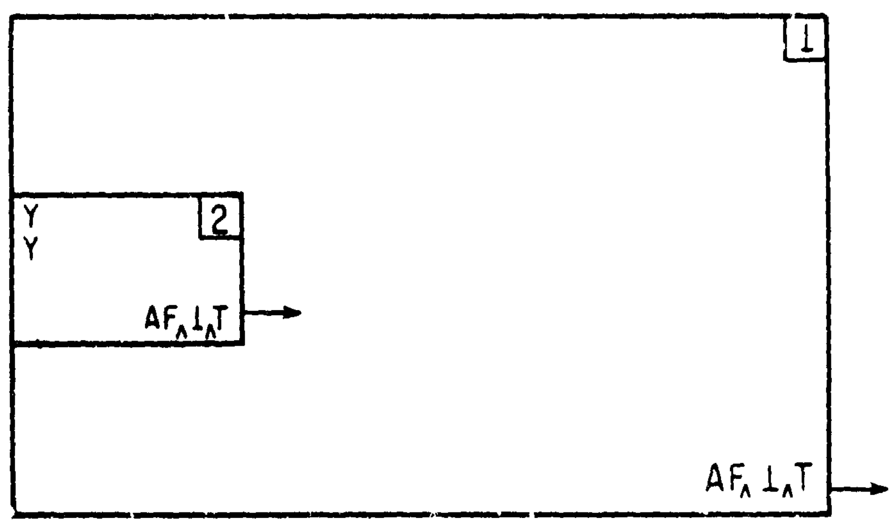


Figure 14(b). Diagramming the SUBSTATE Instruction Using Implied Notation

D. LOGICAL - There are two LOGICAL instructions that allow the user tc logically connect two or more of the transitional instructions together within a single state. These instructions are OR and AND and are written as:

X OR Y or X OR Y OR Z, etc.

X AND Y or X AND Y AND Z, etc.

Where X, Y, and Z are any of the transitional instructions as previously described.

NOTE: The use of LOGICAL instructions requires the use of a state table entry for each element X, Y, or Z and will therefore reduce the maximum number of states from 30 to 30 minus the number of logical instruction elements.

Examples of LOGICAL instruction:

IF 1 R3 OR AF 5 S - exit this state if 1 response 3 occurs or after 5 seconds

IF 1 R 2 AND FOL 1 S 5 - exit this state when 1 response 2 occurs and 1 state 5 has been performed

The LOGICAL instruction is diagrammed in Figure 15 in which state #1 would exit if 1 response 3 occurred or after 5 seconds elapsed.

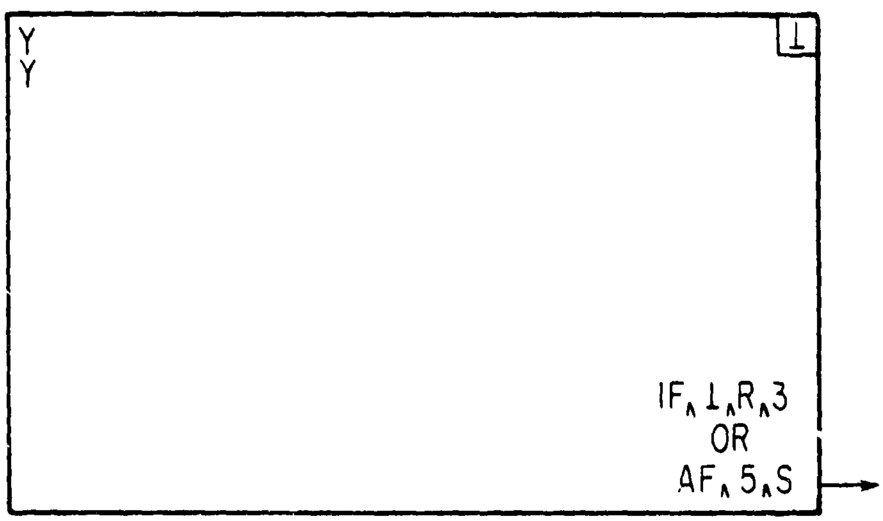


Figure 15. Diagramming the LOGICAL Instruction

E. INPUT/OUTPUT - There are three output instructions that allow output of variables or arrays to the line printer, paper tape punch, and the console CRT. There is one input that allows input of variables or arrays from the paper tape reader. These instructions are PTR, PUN, CRT, and RDR and are written as:

X A

X A; B; C etc.

X D*

Where X is one of the input/output instructions PTR, PUN, CRT, or RDR; A, B, or C is any variable; and D* is the name of any array that has been previously defined using a DIMENSION instruction.

Examples of INPUT/OUTPUT instruction:

PTR A* - print array A on line printer

PUN A - punch variable A on paper tape punch

CRT A; B; C - print variables A, B, and C on console CRT

RDR A* - read array A from paper tape reader

The INPUT/OUTPUT instruction is diagrammed in Figure 16 in which variables A, B, and C are printed on the line printer during state #1.

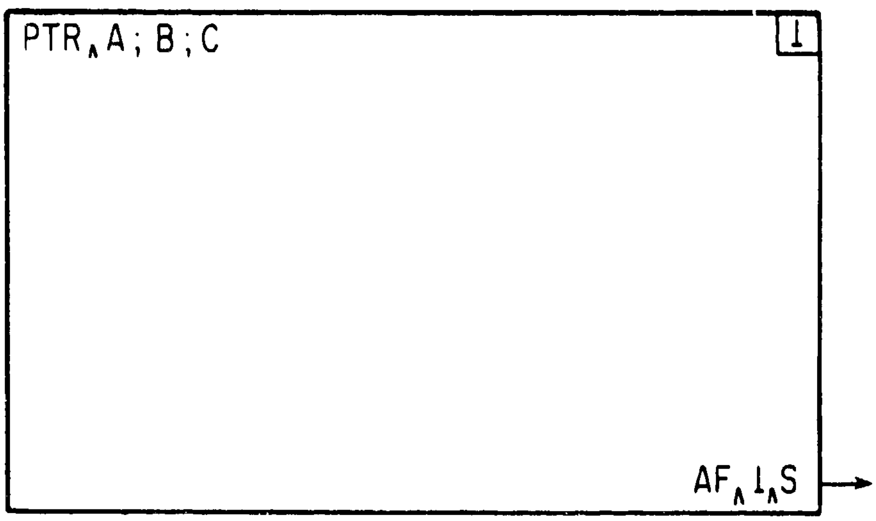


Figure 16. Diagramming the INPUT/OUTPUT Instruction

This concludes the description of the SDS instruction set. It is recommended that Sections IV and V be reviewed prior to proceeding to Section VI which describes the running and use of the SDS. For a list of all available instructions in the SDS refer to Appendix C which describes each set of instructions in detail.

VI. USING THE SDS

A. Introduction to RTE-II

The SDS runs under control of Hewlett-Packard's RTE-II real-time operating system and can be started from either RTE-II or from RTE-II's file manager program FMGR. The procedure used to initialize RTE-II on the HP-2100 is described in detail in Appendix D. When RTE-II is initialized, it automatically schedules program FMGR to be run. PROGRAM FMGR prints the following welcome message on the CRT.

```
SET TIME
:SV,4
TE,*****
TE,*****WELCOME TO THE SDS PLEASE TYPE RU, OPCOM WHEN YOU
TE,*****ARE READY TO BEGIN USING THE  SDS
TE,*****
::
:
```

If the correct time, day, and year is to be maintained by RTE-II the system must be given this information prior to running the operator communications program. To set the real time clock enter the following command:

SYTM, YEAR, DAY, HOUR, MINUTE, SECOND

where:
year is a four digit year.
day is a three digit day of the year i.e., 1 to 365.
hour, minute, second is the current time of a 24-hour clock.

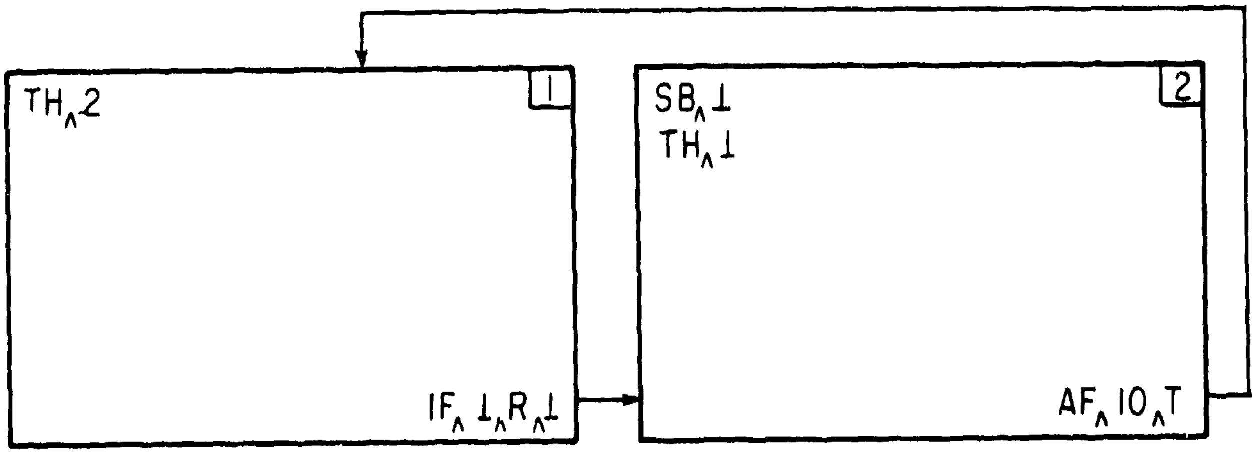
The last character in the above message is a colon (:) which is the prompt character for the FMGR program and indicates that FMGR is ready to accept input from the keyboard. The RTE-II system expects the FMGR program to be used within approximately five minutes. If no use is made of the FMGR program within this time, RTE-II terminates the program. As previously stated, however, the SDS will run under RTE-II or FMGR, and the termination of program FMGR merely changes the method of starting the SDS. If FMGR has not terminated, the user must type RU,OPCOM to start the SDS otherwise he must type *RU,OPCOM.

B. Programming the SDS

The SDS is programmed through the operator communications program OPCOM. When the FMGR command RU,OPCOM is entered on the keyboard, the character @ is printed on the CRT. The @ is the prompt character for the SDS. The required job control instruction NEW$ must then be entered on the keyboard, which tells OPCOM to prepare for entry of a new program. OPCOM then prints the message, "INPUT FROM DISC??", which must be given a yes or no answer. If the reply to this question is yes, OPCOM uses the program contained in disc file OPIN as input, otherwise it expects the operator to input the program from the keyboard. Procedures for creating programs for disc input from file OPIN are described in detail in Appendix E. For the purpose of discussing programming the SDS, it will be assumed that the reply to the above question was no and that the program will be entered through the keyboard. Following each line of input from the keyboard that is properly terminated by the job control instruction $,OPCOM will take action as necessary and then print its prompt character @ indicating that the system is ready for the next line of input.

Since the SDS was written to be used by investigators familiar with schedules of reinforcement, the following pages will use examples of programming the SDS with which they are familiar. These examples show various schedules of reinforcement as described in A Primer of Operant Conditioning, by G. S. Reynolds (1), and include the state diagram and the source language program.

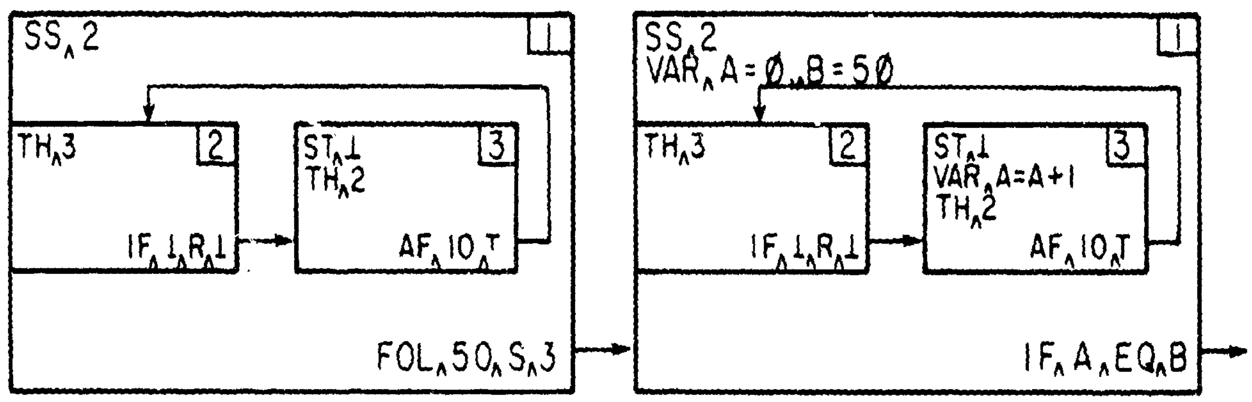
The first example given is that of a continuous reinforcement schedule (CRF) (1, p. 38). Continuous reinforcement is reinforcement that occurs every time a correct response occurs. This procedure would be diagrammed and programmed as shown in Figure 17.




```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 IF 1 R 1 TH 2$
@
ST2 AF 10 T SB 1 TH 1$
@
END$
```

Figure 17. Diagram and Program for a CRF Schedule

If the program in Figure 17 were used, it can be seen that upon every occurrence of response #1 a transition would be made to state #2 in which stimulus #1 would be issued for 10 ticks of the system clock (100 ms). After 100 ms had elapsed a transition would be made back to state #1 where another response #1 would be awaited. A program such as this would tie up the SDS since it has no means of getting out of the continuous loop. Figure 18 shows two ways to exit this continuous loop. The first method uses the following instruction as a counter, and the second method increments a variable during each pass through the loop and tests for the variable being equal to a preset count using the modified IF instruction.

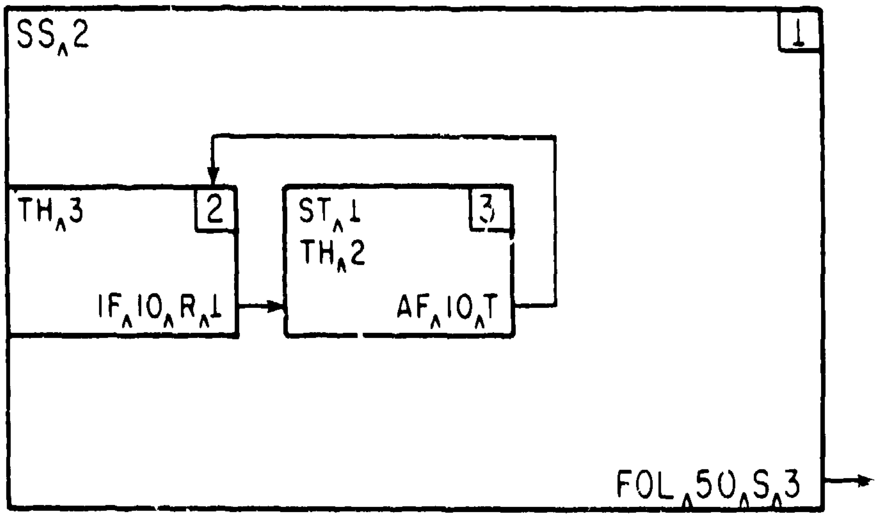



```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 FOL 50 S 3 SS 2$
@
ST2 IF 1 R 1 TH 3$
@
ST3 AF 10 T ST 1 TH 2$
@
END$
```

```
RU,OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 IF A EQ B SS 2 VAR A=Ø, B=50$
@
ST2 IF 1 R 1 TH 3$
@
ST3 AF 10 T ST 1 TH 2 VAR A=A+1$
@
END$
```

Figure 18. Methods of leaving Continuous Loop

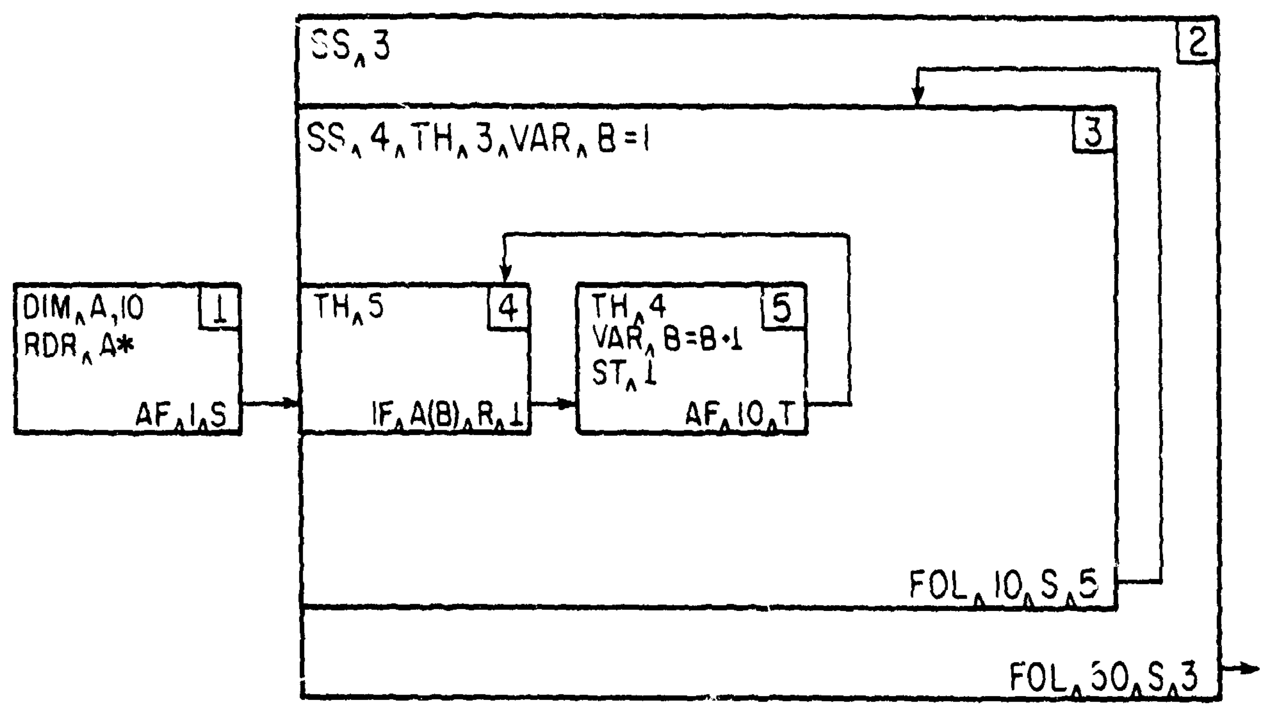
A fixed-ratio schedule (FR) requires that a fixed number of responses be received for every reinforcement (1, p. 67) Figure 19 describes an FR schedule that issues a reinforcement stimulus after each count of 10 response #1. The program will terminate after 50 reinforcements have been issued.




```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 FOL 50 S 3 SS 2$
@
ST2 IF 10 R 1 TH 3$
@
ST3 AF 10 T ST 1 TH 2$
@
END$
```

Figure 19. Diagram and Program for FR Schedule

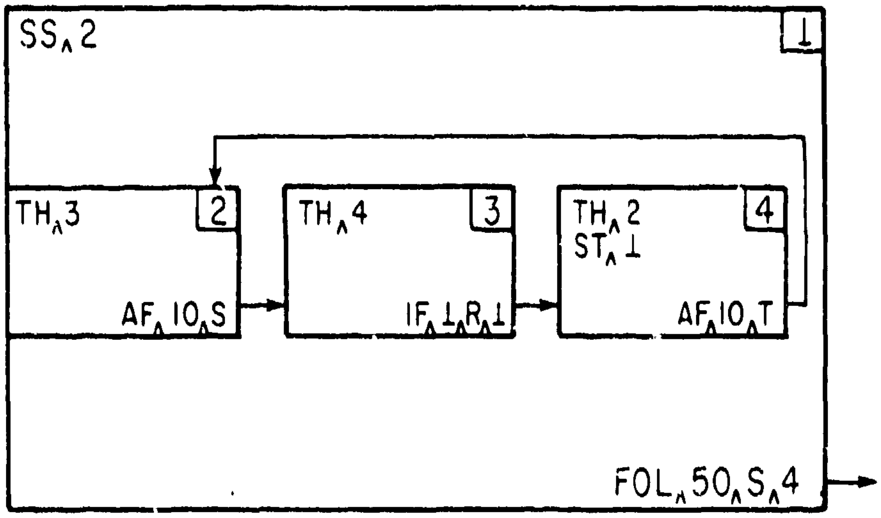
A variable-ratio (VR) schedule is a schedule in which the number of responses required for one reinforcement varies from the number of responses required for other reinforcements (1, p. 67). The number of responses required for each reinforcement are irregular but are usually repeating numbers. Figure 20 describes a VR schedule in which the variable-ratio table is read in from the paper tape reader. When the entire table has been used, the table pointer is reset and the schedule is then repeated. This process is continued until it has been performed 50 times.




```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
ST1 AF 1 S TH 2 DIM A,10 RDR A*$
@
ST2 FOL 50 S 3 SS 3$
@
ST3 FOL 10 S 5 SS 4 TH 3 VAR B=1$
@
ST4 IF A(B) R 1 TH 5$
@
ST5 AF 10 T TH 4 ST 1 VAR B=B+1$
@
END$
```

Figure 20. Diagram and Program for VR Schedule

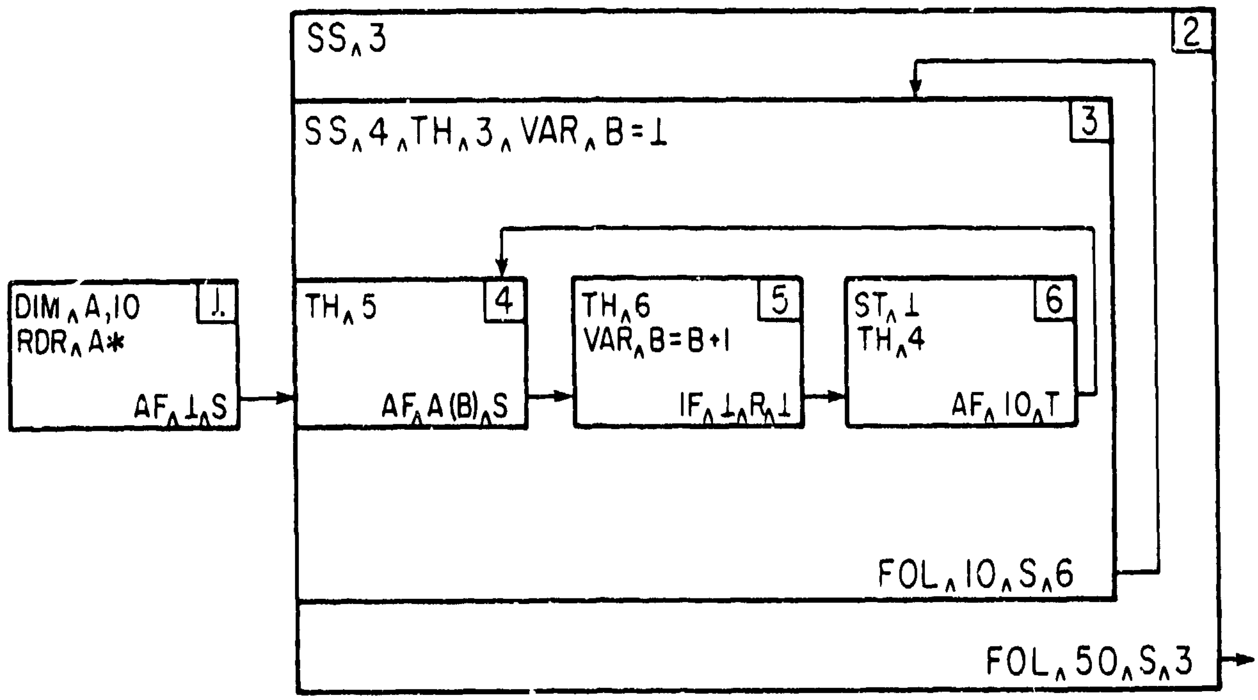
A fixed-interval (FI) schedule has a constant time delay before a response can be reinforced (1, p. 67). Figure 21 describes an FI schedule that issues a reinforcement stimulus after 10 seconds have elapsed if response #1 occurs. The program will terminate after 50 reinforcements have been issued.




```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 FOL 50 S 4 SS 2$
@
ST2 AF 10 S TH 3$
@
ST3 IF 1 R 1 TH 4$
@
ST4 AF 10 T ST 1 TH 2$
@
END$
```

Figure 21. Diagram and Program for FI Schedule

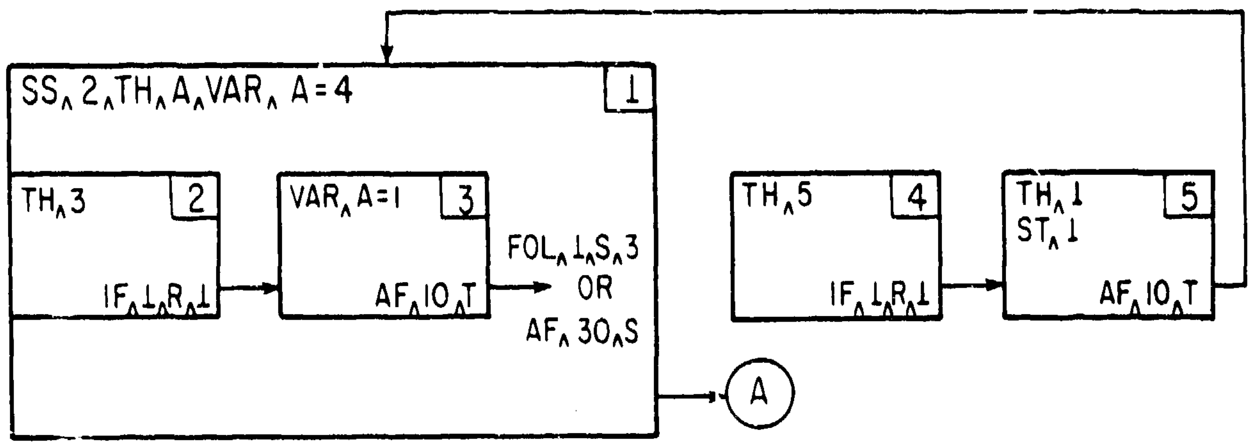
A variable interval (VI) schedule varies the amount of time delay required before a response can be reinforced. Figure 22 describes a VI schedule that issues a reinforcement stimulus, after A(B) seconds have elapsed, if response #1 occurs. The program will terminate after 500 reinforcements have been issued.




```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 AF 1 S TH 2 DIM A, 10 RDR A*$
@
ST2 FOL 50 S 3 SS 3$
@
ST3 FOL 10 S 6 SS 4 TH 3 VAR B=1$
@
ST4 AF A(B) S TH 5$
@
ST5 IF 1 R 1 TH 6 VAR B=B+1$
@
ST6 AF 10 T ST 1 TH 4$
@
END$
```

Figure 22. Diagram and Program for VI Schedule

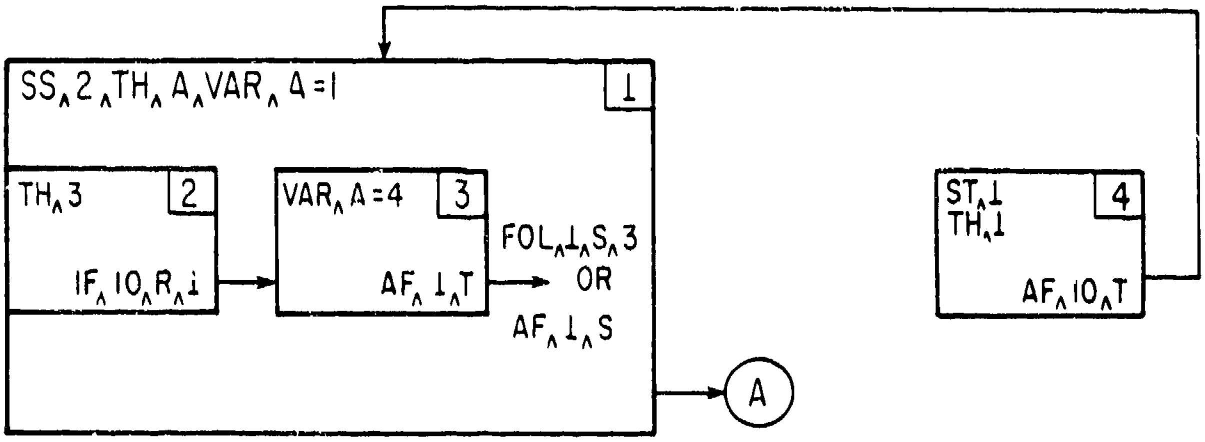
A schedule in which a reinforcement occurs with a response only if a designated amount of time has elapsed since the occurrence of the last response is a differential reinforcement of low rates of responding (DRL) schedule (1, p. 94). Figure 23 describes a DRL schedule in which a reinforcement stimulus is issued if and only if 30 seconds have elapsed since the last response. The technique of using the THEN instruction should be noted since it is an effective means of making conditional branches. No master counter is used in this diagram for simplification of the drawing.




```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 FOL 1 S 3 OR AF 30 S SS 2 TH A VAR A=4$
@
ST2 IF 1 R 1 TH 3$
@
ST3 AF 10 T VAR A=1$
@
ST4 IF 1 R 1 TH 5$
@
ST5 AF 10 T ST 1 TH 1$
@
END$
```

Figure 23. Diagram and Program for DRL Schedule

A schedule in which a reinforcement occurs with a designated number of responses and only if a designated amount of time has not elapsed is a differential reinforcement of high rates of responding (DRH) schedule (1, p. 94). Figure 24 describes a DRH schedule in which a reinforcement stimulus is issued if and only if 10 response #1 occur before one second has elapsed. If one second elapses before the 10 responses occur, the response counter is reset and no reinforcement stimulus is issued. No master counter is used in this diagram for simplification of the drawing.



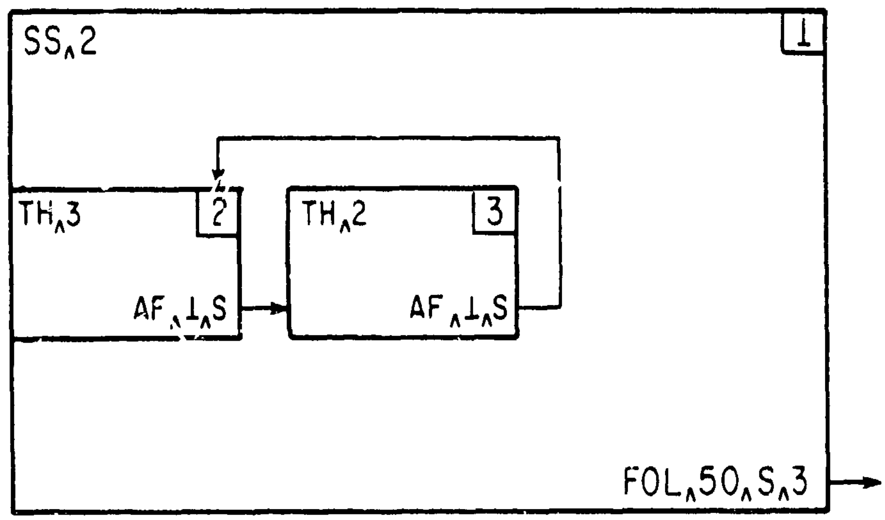

```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 FOL 1 S 3 OR AF 1 S SS 2 TH A VAR A=1$
@
ST2 IF 10 R 1 TH 3$
@
ST3 AF 1 T VAR A=4$
@
ST4 AF 10 T ST 1 TH 1$
@
END$
```

Figure 24. Diagram and Program for DRH Schedule

There are more complex known schedules of positive reinforcement, but since most of these schedules can be reduced to variations of response ratios and time intervals, combined with differential reinforcement of low or high rates of responding when necessary (1,p.67), no other schedules of reinforcement will be discussed in this section. Every form of each instruction was not shown in the preceding examples. Refer to Section V, the SDS Instruction Set, for additional programming and diagramming information. Appendix F describes possible errors that may occur when programming SDS and should be reviewed prior to using the system.

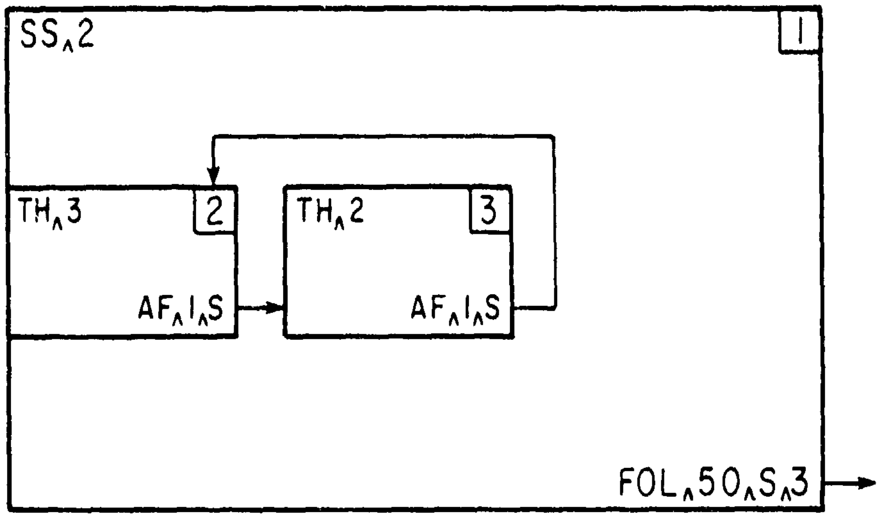
C. Running SDS Programs

Figures 17 through 24, used in describing programming SDS, each include an END$ instruction as the last instruction in the source programs. When this instruction is received by the operator communications package, the message "START EXP.?" is printed on the CRT. A yes or no reply must be entered on the keyboard when this question is asked; otherwise, SDS will wait approximately 32 seconds and assume the answer was yes. If the reply is yes, OPCOM creates an output file named OPOUT that contains a copy of the source program that was input from the console keyboard. OP-COM then instructs SDS to run the experiment. If the reply to the start experiment question was no, OPCOM creates an output file named OPOUT that contains a copy of the source program that was input from the console keyboard. OPCOM then instructs SDS to terminate the program Figures 25 and 26 are examples of use of the yes or no reply given by the user to the question, START EXPERIMENT??. When an SDS program terminates properly, the message "END OF EXPERIMENT 1" is printed on the CRT and the system is suspended. The entry *GO,OPCOM must be entered on the keyboard in order to terminate all programs and return to RTE-II's FMGR.




```
RU,OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 FOL 50 S 3 SS 2$
@
ST2 AF 1 S TH 3$
@
ST3 AF 1 S TH 2$
@
END$
START EXP?
YES
END OF EXP.        1

*GO, OPCOM
```

Figure 25. Programming and Running an SDS Program

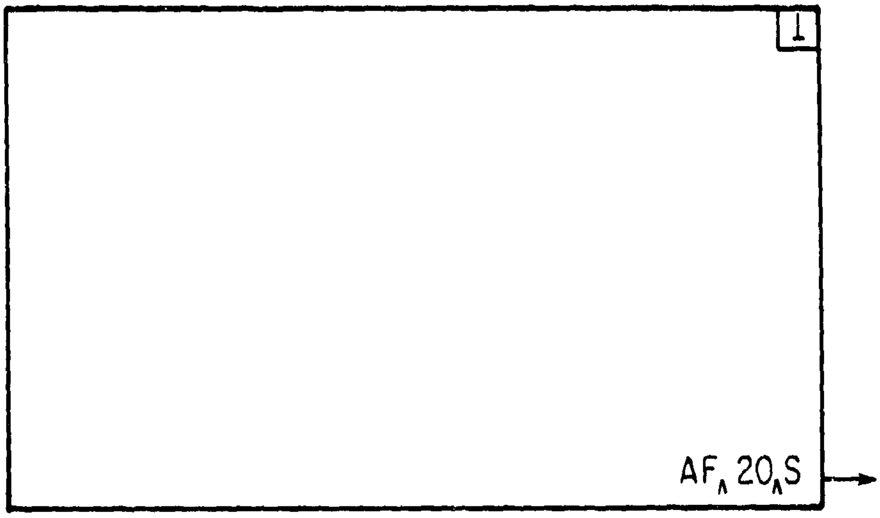



```
RU, OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 FOL 50 S 3 SS 2$
@
ST2 AF 1 S TH 3$
@
ST3 AF 1 S TH 2$
@
END$
START EXP?
NO

*GO, OPCOM
:
```

Figure 26. Programming and Not Running an SDS Program

D. Using Disc Files OPIN and OPOUT

When the operator communications program OPCOM reaches the END$ instruction, as previously mentioned it creates an output file named OPOUT that contains a copy of the source program. OPCOM also moves the file OPOUT into the file OPIN when it terminates the SDS program. This automatic loading occurs to enable multiple runs of the same program without having to retype the program from the console keyboard. Figure 27 shows an example of a multiple run where the first run was input on the keyboard and the second run was input from the disc.




```
:RU,OPCOM
@
NEW$

INPUT FROM DISC??
NO
@
ST1 AF 20 S$
@
END$
START EXP?
YES
END OF EXP          1

*GO, OPCOM
```

```
:RU, OPCOM
@
NEW$

INPUT FROM DISC??
YES
@
NEW$
@
ST1 AF 20 S$
@
END$
START EXP?
YES
END OF EXP.         1

*GO, OPCOM
:
```

Figure 27. Multiple Runs of Program Using Input File OPIN

The following procedure can be used to save the source program in OPOUT for later use. After terminating the SDS issue the following FMGR command:

:ST,OPOUT,NAME

Where

: is the FMGR prompt character not typed by the user;
ST is the FMGR store command;
OPOUT contains the source file;
and NAME is any unused file name selected; i.e., FILE, SAVE, JOB1, etc.

There are several methods to create a file that can be used in OPIN as an input file to OPCOM. These methods are discussed in detail in Appendix E.

VII. THE SDS LOG

The SDS log program is a low priority program that collects data concerning all events. Program LOGG writes the data on magnetic tape when one of its two buffers is filled and at the end of any SDS program. The time of the beginning and end of the experiment, the beginning and end of each state, and the occurrence of all responses is logged on the magnetic tape. This allows the user to determine precisely what events took place during the experiment and to collect statistical data by using off-line programs after the experiment has been completed. The format of each log event is shown in Table III. A sample SDS program run including a copy of the log is described in Appendix G.

Table III

Format of Log for Each Event

Word	Start of Exp.	Start of State	End of State	Resps.	End of State
1	1000	2000	3000	4000	5000
2	Exp. #	Exp. #	Exp. #	Exp. #	Exp. #
3	N/A	State #	State #	State #	N/A
4	N/A	N/A	N/A	Bit #	N/A
5	10s of ms	10s of ms	10s of ms	10s of ms	10s of ms
6	seconds	seconds	seconds	seconds	seconds
7	minutes	minutes	minutes	minutes	minutes
8	hours	hours	hours	hours	hours
9	day of year	day of year	day of year	day of year	day of year

NOTE: Word two of each event, the experiment number, was included to facilitate future expansion of the SDS system to control multiple experiments. Words five thru nine give precise time of day and day of year only if the proper time and date were set up using RTE commands; otherwise, they are relative to the initial values of words five thru nine.

VIII. REFERENCE

1. Reynolds, G. S., A Primer of Operant Conditioning. Second Edition. Glenview. Illinois: Scott, Foresman and Company, 1975.

APPENDIX A

IN-CORE MULTITASKING USING RTE-II

APPENDIX A

IN-CORE MULTITASKING USING RTE-II

SYSTEM GENERATION

The approach used to accomplish the desired multitasking capabilities required that a new system be generated that included eight dummy programs. These programs were named T1XXX through T8XXX. These names were selected to identify them as tasks (T), to number them (1-8), and to meet the requirement of the event sense interface routine that the last three letters be Xs. Since these programs are included only for the purpose of generating ID segments and are never executed, their contents can be any valid instructions. It is important to keep them as short as possible because they are foreground resident programs and therefore are wasted core. The programs used in this generation were all identical except for the NAM statement and included the following code:

```
ASMB,R,L
          NAM T1XXX,1,4
BGN1      HLT
          END BGN1
```

Each of these programs required only one word of core and is non-executable due to the HLT instruction. If the programs were scheduled before their ID segments were modified, as discussed later in this Appendix, a memory protect violation would occur in the system which serves as a desirable warning.

1) Generating these tasks into the system requires several entries in the answer file for the on-line generator, RTGN2. These entries are:

Program input phase

```
REL,%T1XXX
REL,%T2XXX
REL,%T3XXX
REL,%T4XXX
REL,%T5XXX
REL,%T6XXX
REL,%T7XXX
REL,%T8XXX
REL,%NTSK
```

2) The interrupt table included entries assigned to unused device select codes for future modifications and uses, such as driver debugging and for setting up pointers to the desired ID segments.

```
36,PRG,T1XXX
37,PRG,T2XXX
40,PRG,T3XXX
41,PRG,T4XXX
42,PRG,T5XXX
43,PRG,T6XXX
44,PRG,T7XXX
45,PRG,T8XXX
```

Upon completion of the system generation three goals have been achieved. First, the dummy programs are in core and will issue a memory protect warning when used improperly. Secondly, an ID segment has been generated for each of the dummy programs which will be used to load multiple programs into core using the RTE-II loader. Each of these programs required one word of foreground core, two words in the device reference table, one word in the interrupt table, and 22 words for the ID segment. Third, a pointer to each of the programs has been set up in the interrupt table.

In order to describe the use of the multitasking technique, two programs will be used. These programs named PROGA and PROGB contain the following code.

```
FTN4,L
     PROGRAM PROGA
     DIMENSION NAM(3)
     DATA NAM/2HPR,2HOG,2HB /
     WRITE (1,10)
10 FORMAT ("TEST PROGA")
     CALL EXEC (9,NAM)
     END
     END$
FTN4,L
     PROGRAM PROGB
     WRITE (1,10)
10 FORMAT ("TEST PROGB")
     END
     END$
```

Assuming these programs had been compiled and relocatable programs were available, any attempt to load and run them in-core in a single partition using RTE-II's loader would fail. For example, if a procedure was used as follows:

```
:LG,1
:MR,%PROGA
:MR,%PROGB
:RU,LOADR,99,1
```

The loader would accept both relocatable programs, load them into background, show both of them on the load map, but allow only PROGA to be run. On the other hand, if the two programs were loaded separately, the programs would load and run but would not be contained in core simultaneously.

If program PROGB were modified and made a subroutine, as shown below, both PROGA and PROGB would be in core simultaneously, but when PROGA attempted to execute the CALL EXEC (9,NAM) an error would occur because PROGB would have no ID segment. Getting both programs into core simultaneously is necessary; therefore, PROGB would need to be modified as follows:

```
FTN4,L
          SUBROUTINE PROGB
          WRITE(1,10)
10        FORMAT ("TEST PROGB")
          END
          END$
```

In order to have the capability of scheduling PROGB, an ID segment generated for one of the dummy programs will now be modified to point to PROGB. To accomplish this PROGA must be modified as follows:

```
FTN4,L
          PROGRAM PROGA
          DIMENSION NAM(3)
          DATA NAM/2HT1,2HXX,2HX /
          WRITE(1,10)
10        FORMAT("TEST PROGA")
          CALL NTSK1
          CALL EXEC (9,NAM)
          END
          END$
```

The call to subroutine NTSK1, discussed later in this Appendix, would modify program T1XXX's ID segment to point to PROGB. PROGB can now be scheduled using the CALL EXEC (9,NAM); however, it must now be referred to as T1XXX. Using this technique as many as nine programs can be in a single partition simultaneously. In this case, for example, PROGA and T1XXX THROUGH T8XXX could be installed in a single partition, removing the necessity to swap programs to and from the disk in order to multiprogram. This approach was desirable in the State Diagram System primarily because of the overhead in swapping programs in and out of core.

There were two options available when attempting to modify ID segments using subroutine NTSK1. The first was to contain the code required to modify the ID segment and the code to establish the pointer to the program being installed, PROGB in the above example, in a single subroutine. The second option was to contain the code required to modify the ID segment in one subroutine and to have as many as eight subroutines to generate pointers to programs being installed. Since the latter method consumed less core, it was selected for use in SDS.

The following subroutine was generated to establish pointers to those programs being installed in core simultaneously.

```
ASMB,R,L
              NAM NTSK1,6
              EXT NTSK,PROGB,.ENTR
              ENT NTSK1
NTSK1         NOP
              JSB .ENTR
              NOP
              JSB NTSK
              DEF *+3
              DEF ONE
              DEF PROGB
              JMP NTSK1,I
ONE           OCT 1
              END
              END$
```

Subroutine NTSK1 gets the entry point address of the program being installed, PROGB in the above example, and passes it to subroutine NTSK for installation in the ID segment of T1XXX as designated by the first parameter labeled ONE. As can be seen, a similar subroutine would be required to install other programs. For example, by changing all NTSK1 to NTSK2 and replacing those underlined words in the program with PROGC, TWO, PROGC, TWO, and 2, respectively, the subroutine would install PROGC in core using T2XXX's ID segment.

The following subroutine was generated to modify the ID segments of T1XXX through T8XXX. It can be included in the system during system generation or it can be loaded using RTE-II's loader.

```
ASMB,R,L
              NAM NTSK,6
              ENT NTSK
              EXT $LIBR,$LIBX,.ENTR,EXEC
ARG           BSS 2
NTSK          NOP
              JSB .ENTR
              DEF ARG
              LDB ARG,I       GET TASK NUMBER
              ADB TABA        SET UP INDEX
              JSB $LIBR       DROP FENCE
              NOP
```

```
          LDA BREG,I       GET INT TABLE POSIT
          ADA INTBA        SET UP INDEX
          LDB, AREG, I     GET ID SEG ADR
          SSB,RSS          NEGATIVE?
          JMP ADONE         NO
          CMB,INB           YES MAKE IT POS.
          STB AREG,I        AND PUT IN INT TABLE
ADONE     ADB B7            ADJ TO WORD 8
          LDA ARG+1         GET TASK ADR
          ADA =B3           ENTRY POINT
          STA BREG,I        INSTALL TASK
          JSB $LIBX         UP FENCE
          DEF NTSK
          JMP NTSK,I        RETURN
TABA      DEF *
          OCT 36
          OCT 37
          OCT 40
          OCT 41
          OCT 42
          OCT 43
          OCT 44
          OCT 45
AREG      EQU OB
BREG      EQU 1B
INTBA     EQU 1654B
B7        OCT 7
          END
          END$
```

Subroutine NTSK receives the desired task number and task entry point from subroutines NTSK1-NTSK8. This information is then used to modify word eight of the desired ID segment, making it point to the task being installed. For the State Diagram System no attempt was made to change the name words in the ID segment. If calling PROGB, T1XXX, after the task installation is undesirable words 13, 14, and 15 of the ID segment must be modified to contain PR, OG, and B߸, respectively. This would not be a difficult task and subroutine NTSK could easily be modified to attain this goal.

APPENDIX B

SDS PROGRAM AND SUBROUTINE LISTINGS

"OPCOM T=00003 IS ON CR00002 USING 00024 BLKS R=0000

```
FTN4
      PROGRAM OPCOM
      COMMON J,IARAY(72),IVT(90),ISH,IV,NBR1
      COMMON NBR2,NBR3,NBR01,IREL,ITERM,KV
      COMMON LINE(15),IAFLG(5),INDAY(15)
      DIMENSION NAM(3),IPRM(5),IANS(2),NAM1(3)
      DIMENSION IB1(1),IB2(36)
      DIMENSION IB7(10),IB8(12),IB9(14),IB10(12),IB11(14)
      DIMENSION IB15(18),NAM2(3),IRB(3)
      EQUIVALENCE (IRB(1),NBR1),(IRB(2),NBR2),(IRB(3),NBR3)
      DATA NAM/2HT4,2HXX,2HX /
      DATA NAM1/2HSD,2HS ,2H  /
      DATA NAM2/2HT6,2HXX,2HX /
      DATA IB1/2H @/
      DATA IB7/2HST,2HAR,2HT ,2HEX,2HP?/
      DATA IB8/2HBA,2HD ,2HOP,2HER,2HAT,2HOR/
      DATA IB9/2HOU,2HTS,2HID,2HE ,2HTA,2HBL,2HE /
      DATA IB10/2HNO,2H O,2HCT,2HAL,2H N,2HBR/
      DATA IB11/2HBA,2HD ,2HBI,2HNA,2HRY,2H F,2HMT/
      DATA IB15/2HIN,2HPU,2HT ,2HFR,2HOM,2H D,2HIS,2HK ,2H??/
C INITIALIZE VARIABLES
      J=0
      IDISK=0
      ISH=0
      IREL=0
      INEW=0
      DO 1 I=1,5
1     IAFLG(I)=0
C ENTRY POINT FOR NEW STATE
6     IANOR=0
      IFLG=0
      DO 5 I=1,15
5     LINE(I)=0
      DO 3 I=1,90
3     IVT(I)=0
      NTERM=0
C KV IS VAR TABLE PTR
      KV=1
C ENTRY POINT TO CONTINUE A STATE
7     ITERM=0
C PROMPT OPERATOR TO INPUT NEXT LINE
      CALL EXEC(2,1,IB1,1)
C READ A LINE AND POSITION PROPERLY
C IF DISK FLAG IS SET READ FROM DISK
      IF(IDISK.NE.1)GO TO 11
      IP=0
C SKED RDWRT FOR READ
      CALL EXEC(9,NAM2,IP)
C GET CLASS NUMBER THEN READ A RECORD
      CALL RMPAR(IPRM)
```

```
      CALL EXEC(21,IPRM(2),IB2,36)
      DO 8 I=1,36
      ICOND=IAND(IB2(I),177400B)
      IF(ICOND.EQ.22000B)IB2(I)=22040B
      IF(ICOND.EQ.22000B)IBK=I+1
      IF(ICOND.EQ.22000B)GO TO 87
      ICOND=IAND(IB2(I),377B)
      IF(ICOND.EQ.44B)IBK=I+1
      IF(ICOND.EQ.44B)GO TO 87
8     CONTINUE
      GO TO 89
87    DO 88 I=IBK,36
88    IB2(I)=20040B
C WRITE RECORD ON CRT
89    CALL EXEC(2,1,IB2,36)
      GO TO 13
C READ RECORD FROM KEYBOARD
1     CALL EXEC(1,401B,IB2,-72)
C IF NEW FLAG IS CLEAR RDWRT TASK IS NOT INSTALLED
13    IF(INEW.NE.1)GO TO 12
      IP1=1
      IP=1
      ICLAS=0
C SEND A RECORD TO RDWRT
      CALL EXEC(20,0,IB2,36,JDUM,IDUM,ICLAS)
C DO NOT WRITE NEW$ RECORD
      IF(IB2(1).EQ.2HNE)GO TO 12
C SKED RDWRT FOR WRITE
      CALL EXEC(9,NAM2,IP,IP1,ICLAS)
C CONDX STATE RECORD AND PUT IN IARAY
12    J1=1
      DO 9 I=1,36
      IARAY(J1)=IAND(IB2(I),177400B)
      IARAY(J1)=ISHFT(IARAY(J1),-8)
      IARAY(J1+1)=IAND(IB2(I),377B)
9     J1=J1+2
      J=0
C TERMINATING CHAR IN LINE ?
      DO 10 I=1,72
      IF(IARAY(I).EQ.44B) J=J+1
      IF(IARAY(I).EQ.44B) GO TO 15
10    CONTINUE
C NO TERMINATING CHAR IN LINE THEN SET FLAG
      NTERM=1
      CALL SKPSP
      IF(ITERM.EQ.2)GO TO 7
C NEW ?
15    IF(IARAY(J).EQ.116B) GO TO 70
C END ?
      IF(IARAY(J).EQ.105B) GO TO 85
C STATE ?
17    IF(IARAY(J).NE.123B) GO TO 20
C STATE LOGIC
      CALL SKPSP
      IF (ITERM.NE.0) GO TO 1020
C IF STATE FLAG IS SET THIS MUST BE SUBSTATE OR STIMULUS
```

```
      IF(IFLG.EQ.1)GO TO 451
      IF (IARAY(J).NE.124B) GO TO 451
      CALL SKTNB
      IF(ITERM.NE.0) GO TO 1020
      IANOR=0
      IF(NTERM.EQ.1)IFLG=1
      CALL NBRTY
      IF(ITERM.NE.0)GO TO 1020
      ISN=NBR2
C STATE NUMBER WITHIN LIMITS ?
      IF(ISN.GT.30) GO TO 802
      GO TO 30
20    IF(NTERM.NE.1) GO TO 802
C IF ?
30    IF(IARAY(J).EQ.111B)GO TO 100
C AFTER ?
      IF(IARAY(J).EQ.101B)GO TO 200
C FOLLOWING ?
      IF(IARAY(J).EQ.106B) GO TO 300
C AND/OR ?
      IF(IANOR.NE.1) GO TO 400
      J=J-2
      GO TO(103,203,303,103),LINE(1)
C GO TO THEN LOGIC SINCE NONE OF ABOVE WAS TRUE
C NEW LOGIC
70    CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.105B) GO TO 800
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.127B) GO TO 800
      IPRM(1)=1
C NEW OK CALL SDS TO CLEAN HOUSE AND INSTALL TASKS
C DO NOT PERFORM NEW$ TWICE
      IF(INEW.EQ.1)GO TO 6
      CALL EXEC(9,NAM1,IPRM(1))
      IV=1
C SET NEW FLAG
      INEW=1
      IP1=0
      IP=1
      ICLAS=0
      CALL EXEC(20,0,IB2,36,JDUM,IDUM,ICLAS)
C SKED RDWRT FOR NEW$ WRITE
      CALL EXEC(9,NAM2,IP,IP1,ICLAS)
      CALL EXEC(3,1101B,1)
      CALL EXEC(2,1,IB15,18)
      CALL EXEC(1.401B,IANS,2)
C SET DISK FLAG IF READ FROM DISK
      IF(IANS.EQ.2HYE)IDISK=1
      GO TO 6
C END LOGIC
85    CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.116B) GO TO 800
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
```

```
      IF(IARAY(J).NE.104B) GO TO 800
C END OK CALL SDS WITH NECESSARY PARMS
      CALL EXEC(2,1,IB7,10)
      CALL EXEC(1,401B,IANS,2)
C START EXPERIMENT ?
      IF(IANS.EQ.2HNO)GO TO 92
C SET START FLAG
      ISTRT=1
      IPRM(1)=1
      IPRM(2)=1
      IPRM(3)=0
      IPRM(4)=1
      IPRM(5)=0
      CALL EXEC(10,NAM1,IPRM(1),IPRM(2),IPRM(3),IPRM(4),IPRM(5))
92    CALL EXEC(7)
      CALL EXEC(3,1101B,1)
      IP=3
      CALL EXEC(9,NAM2,IP)
      IF(IANS.NE.2HNO)CALL EXEC(6,NAM1)
      CALL EXEC(6)
      GO TO 9999
C ********************************************************* IF LOGIC
C IF LOGIC
100   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.106B) GO TO 790
      IF(IANOR.NE.0) GO TO 103
C TYPE=1
      LINE(1)=1
      GO TO 105
103   IVT(KV)=1
      KV=KV+3
105   CALL SKP1
      IF(ITERM.NE.0)GO TO 1020
C GET COUNT
      CALL NBRTY
      IF(ITERM.NE.0)GO TO 1020
      IF(IANOR.NE.0) GO TO 108
C SET UP COUNT
      LINE(4)=NBR1
      LINE(5)=NBR2
      LINE(6)=NBR3
      GO TO 110
108   CALL SVARA
C RELATIONAL FUNCTION ?
110   JFLG=J
      IF(IARAY(J).NE.122B) GO TO 175
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
C NO - MULTIPLE RESPONSE ?
      IF(JFLG-(J-1).NE.0)GO TO 112
      IF(IARAY(J).EQ.102B)LINE(1)=-1
      IF(IARAY(J).EQ.102B) CALL SKPSP
C NO - GET RESPONSE NUMBER
112   CALL NBRTY
      IF(IANOR.NE.0) GO TO 113
C SET UP OPERAND
```

```
      LINE(7)=NBR1
      LINE(8)=NBR2
      LINE(9)=NBR3
      GO TO 115
113   CALL SVARA
      IV=IV+9
115   KK=1
      GO TO 1010
C OR ?
611   IF(IARAY(J).NE.117B) GO TO 135
C OR LOGIC
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.122B) GO TO 800
      IF(IANOR.NE.0) GO TO 120
C SET UP OR TYPE AND PTR
      LINE(2)=2
118   LINE(3)=IV
      IANOR=1
130   IVT(KV+1)=0
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      GO TO 30
120   KV=KV-8
      IVT(KV)=2
125   KV=KV+1
      IVT(KV)=IV
      KV=KV+7
      GO TO 130
C AND ?
135   IF(IARAY(J).NE.101B) GO TO 400
C AND LOGIC
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.116B) GO TO 800
      CALL SKP1
      IF(ITERM.NE.0)GO TO 1020
      J=J-1
      IF(IANOR.NE.0) GO TO 140
C SET UP AND TYPE AND PTR
      LINE(2)=1
      GO TO 118
140   KV=KV-8
      IVT(KV)=1
      GO TO 125
C MULTIPLE RESPONSE LOGIC
145   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      J=J-1
      IRB(1)=0
      IRB(2)=0
      IRB(3)=0
      I=0
      IF(IANOR.NE.0) GO TO 147
C SET UP TYPE
      LINE(1)=-1
      GO TO 150
147   KV=KV-6
```

```
      IVT(KV)=-1
      KV=KV+6
150   I=I+1
151   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      DO 153 K=1,8
      IF(IARAY(J).EQ.(47+K)) GO TO 155
153   CONTINUE
      GO TO 160
155   CALL NBRO
      IRB(I)=NBRO1
160   IF(IARAY(J).NE.54B) GO TO 164
      IF(I.GE.3) GO TO 806
      GO TO 150
164   IF(I.LT.3) GO TO 806
      J=J-1
165   IF(IANOR.NE.0)GO TO 167
C SET UP OPERAND
      LINE(7)=IRB(1)
      LINE(8)=IRB(2)
      LINE(9)=IRB(3)
      GO TO 170
166   IF(IANOR.NE.0)GO TO 167
      LINE(4)=NBR1
      LINE(5)=NBR2
      LINE(6)=NBR3
      GO TO 170
167   CALL SVARA
      IV=IV+9
170   CALL SKPSP
      KK=2
      GO TO 1010
C EQUAL ?
175   IF(IARAY(J).EQ.105B)GO TO 180
C NOT EQUAL ?
      IF(IARAY(J).EQ.116B) GO TO 184
C LESS THAN ? LESS THAN OR EQUAL ?
      IF(IARAY(J).EQ.114B) GO TO 187
C GREATER THAN ? GREATER THAN OR EQUAL ?
      IF(IARAY(J).NE.107B) GO TO 800
177   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
C GREATER THAN ?
      IF(IARAY(J).EQ.124B) GO TO 179
C GREATER THAN OR EQUAL ?
      IF(IARAY(J).NE.105B)GO TO 800
      IREL=8
      GO TO 190
179   IREL=6
      GO TO 190
C EQUAL LOGIC
180   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.121B) GO TO 800
      IREL=4
      GO TO 190
C NOT EQUAL LOGIC
```

```
184   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.105B) GO TO 800
      IREL=9
      GO TO 190
187   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
C LESS THAN ?
      IF(IARAY(J).EQ.124B) GO TO 189
C LESS THAN OR EQUAL ?
      IF(IARAY(J).NE.105B) GO TO 800
      IREL=7
      GO TO 190
189   IREL=5
      GO TO 190
C SET UP RELATIONAL TYPE
 190  IF(IANOR.NE.0)GO TO 191
      LINE(1)=IREL
      GO TO 192
191   IVT(KV-6)=IREL
192   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      CALL NBRTY
C SET UP OPERAND
      IF(IANOR.NE.0)GO TO 193
      LINE(7)= NBR1
      LINE(8)= NBR2
      LINE(9)=NBR3
      GO TO 194
193   CALL SVARA
      IV=IV+9
194   KK=4
      GO TO 1010
C ************************************************** AFTER
C AFTER LOGIC
200   CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.106B) GO TO 790
      IF(IANOR.NE.0) GO TO 203
C SET UP AFTER TYPE
      LINE(1)=2
      GO TO 205
203   IVT(KV)=2
      KV=KV+3
205   CALL SKP1
      IF(ITERM.NE.0)GO TO 1020
      CALL NBRTY
      IF(NBR1.NE.1) GO TO 166
      IF(ITERM.NE.0)GO TO 1020
C TICKS ?
      IF(IARAY(J).NE.124B) GO TO 213
      NBR2=-NBR2
      GO TO 166
C SECONDS ?
213   IF(IARAY(J).EQ.123B) GO TO 166
C MINUTES ?
      IF(IARAY(J).NE.115B) GO TO 220
```

```
        NBR2=NBR2*60
        GO TO 166
C HOURS ?
220     IF(IARAY(J).NE.110B) GO TO 800
        NBR2=NBR2*3600
        GO TO 166
C ********************************************** FOLLOWING
C FOLLOWING LOGIC
300     CALL SKPSP
        IF(ITERM.NE.0)GO TO 1020
        IF(IARAY(J).NE.117B) GO TO 790
        IF(IANOR.NE.0) GO TO 303
C SET UP FOLLOWING TYPE
        LINE(1)=3
        GO TO 305
303     IVT(KV)=3
        KV=KV+3
305     CALL SKP1
        IF(ITERM.NE.0)GO TO 1020
        CALL NBRTY
        IF(ITERM.NE.0)GO TO 1020
        IF(IANOR.NE.0) GO TO 308
C SET UP COUNT
        LINE(4)=NBR1
        LINE(5)=NBR2
        LINE(6)=NBR3
        GO TO 632
308     CALL SVARA
C FOLLOWING STATE ?
632     IF(IARAY(J).NE.123B) GO TO 800
        CALL SKP1
        IF(ITERM.NE.0)GO TO 1020
        CALL NBRTY
C YES GO SET UP OPERAND
        J=J-1
        GO TO 165
C THEN ? SUBSTATE ?
C ********************************************** THEN
400     IF(IARAY(J).NE.124B) GO TO 45
C THEN LOGIC
        CALL SKPSP
        IF(ITERM.NE.0)GO TO 1020
        IF(IARAY(J).NE.110B) GO TO 800
        CALL SKP1
        IF(ITERM.NE.0)GO TO 1020
        CALL NBRTY
C SET UP NEXT STATE
        LINE(10)=NBR1
        LINE(11)=NBR2
        LINE(12)=NBR3
        KK=3
        GO TO 1010
C SUBSTATE ? STIMULUS ?
450     IF(IARAY(J).NE.123B) GO TO 900
        CALL SKPSP
        IF(ITERM.NE.0)GO TO 1020
431     IF(IARAY(J).EQ.124B) GO TO 500
```

```
      IF(IARAY(J).EQ.102B) GO TO 500
C SUBSTATE LOGIC
452   IF(IARAY(J).EQ.125B) GO TO 455
C CAN BE WRITTEN AS SUBSTATE OR SS SO CK IT
      IF(IARAY(J).NE.123B)GO TO 800
455   CALL SKP1
      IF(ITERM.NE.0)GO TO 1020
      CALL NBRTY
C SET UP SUBSTATE NUMBER
      IF(NBR2.GT.30)GO TO 802
      LINE(13)=NBR2
      KK=4
      GO TO 1010
C STIMULUS LOGIC
500   CALL SKPSP
      CALL NBRTY
      J=J-1
505   IPRM(1)=3
      IPRM(2)=IRB(1)
      IPRM(3)=IRB(2)
      IPRM(4)=IRB(3)
      IPRM(5)=ISH
      CALL EXEC(9,NAM,IPRM(1),IPRM(2),IPRM(3),IPRM(4),IPRM(5))
      CALL SKPSP
      KK=5
      GO TO 1010
C INITIALIZE VARIABLE LOGIC
900   IF(IARAY(J).NE.126B) GO TO 950
      LINE(14)=1
      LINE(15)=IV
901   CALL SKP1
      IF(ITERM.NE.0) GO TO 1020
      CALL NBRTY
      IF(ITERM.NE.0) GO TO 1020
      KV=KV+3
      CALL SVARA
      J=J-1
      CALL SKPSP
      IF(ITERM.NE.0) GO TO 1020
      IF(IARAY(J).NE.75B) GO TO 800
      CALL SKPSP
      IF(ITERM.NE.0) GO TO 1020
      IF(IARAY(J).EQ.55B)IVT(KV-6)=-1
      IF(IARAY(J).EQ.55B)J=J+1
      IF(IARAY(J).EQ.53B)J=J+1
      CALL NBRTY
      CALL SVARA
      IF(NBR1.EQ.1)GO TO 902
      IF(IVT(KV-9).EQ.-1)IVT(KV-3)=-NBR1
      GO TO 905
902   IF(IVT(KV-9).EQ.-1)IVT(KV-2)=-NBR2
905   IVT(KV-9)=9
      IV=IV+9
      KK=6
      GO TO 1010
903   IF(IARAY(J).EQ.54B) GO TO 910
      IF(IARAY(J).EQ.53B)J=J+1
      IF(IARAY(J-1).EQ.53B) GO TO 904
```

```
      IF(IARAY(J).NE.55B) GO TO 400
      J=J+1
      IVT(KV-9)=-1
904   CALL NBRTY
      CALL SVARA
      IF(NBR1.EQ.1) GO TO 906
      IF(IVT(KV-12).EQ.-1) IVT(KV-3)=-NBR1
      GO TO 908
906   IF(IVT(KV-12).EQ.-1)IVT(KV-2)=-NBR2
908   IVT(KV-12)=12
      IV=IV+3
      KK=7
      GO TO 1010
907   IF(IARAY(J).NE.54B) GO TO 400
      IVT(KV-11)=1
      IVT(KV-10)=IV
909   J=J-1
      GO TO 901
910   IVT(KV-8)=1
      IVT(KV-7)=IV
      GO TO 909
C INITIALIZE ARRAY LOGIC
950   IF(IARAY(J).NE.104B)GO TO 960
      CALL SKP1
      IF(ITERM.NE.0)GO TO 1020
      DO 951 I=1,5
      IF(IAFLG(I).EQ.0)GO TO 952
951   CONTINUE
      GO TO 802
952   IAFLG(I)=IARAY(J)
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IF(IARAY(J).NE.54B)ITERM=1
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      CALL NBRTY
      NWDS=NBR2
      I=I*3
      IF(I.EQ.3)INDAY(I)=27
      IF(I.NE.3)INDAY(I)=INDAY(I-5)+1
      INDAY(I-1)=NWDS
      INDAY(I-2)=INDAY(I)+INDAY(I-1)-1
      IF(INDAY(I-2).GT.226)GO TO 802
      KK=7
      GO TO 1010
C OUTPUT TO CRT ?
960   IF(IARAY(J).NE.103B)GO TO 965
C TYPE 2 IS CRT OUTPUT
      ITYPE=2
C SET UP VAR TABLE FOR INPUT/OUTPUT
963   CALL SKP1
      IF(ITERM.NE.0)GO TO 1020
      IVT(KV)=6
      IVT(KV+1)=ITYPE
      IVT(KV+2)=0
      LINE(14)=ITYPE
      LINE(15)=IV
962   CALL NBRTY
```

```
      KV=KV+3
      CALL SVARA
      IV=IV+6
      KK=8
      GO TO 1010
961   IF(IARAY(J).NE.73B)GO TO 400
      CALL SKPSP
      IF(ITERM.NE.0)GO TO 1020
      IVT(KV)=6
      IVT(KV+1)=ITYPE
      IVT(KV+2)=0
      IVT(KV-4)=IV
      GO TO 962
C INPUT FROM READER ?
965   IF(IARAY(J).NE.122B)GO TO 967
      ITYPE=3
      GO TO 963
C OUTPUT TO PUNCH OR PRINTER ?
967   IF(IARAY(J).NE.120B)GO TO 800
C OUTPUT TO PUNCH ?
      J=J+1
      IF(IARAY(J).NE.125B)GO TO 969
      ITYPE=4
      GO TO 963
C OUTPUT TO PRINTER ?
969   IF(IARAY(J).NE.124B)GO TO 800
      ITYPE=5
      GO TO 963
790   IF(IANOR.EQ.0)GO TO 800
      J=J-2
      GO TO (103,203,303,103),LINE(1)
C INCORRECT OPERATOR
800   CALL EXEC(2,1,IB8,12)
      GO TO 6
C OUTSIDE TABLE
802   CALL EXEC(2,1,IB9,14)
      GO TO 6
C NO OCTAL NUMBER
804   CALL EXEC(2,1,IB10,12)
      GO TO 6
C INCORRECT BINARY FORMAT
806   CALL EXEC(2,1,IB11,14)
      GO TO 6
C CALL DOOPS TO CHANGE A SINGLE STATE
1000  IF(ISN.EQ.0) GO TO 6
C VAR TABLE FLAG CK'D HERE ALSO EX 20 AND DOOPS IF NEEDED FOR IVAR
      IF(KV.EQ.1)GO TO 1005
      ICLAS=0
      IPRM(1)=50
      IPRM(2)=KV-1
      IPRM(3)=IV-IPRM(2)
      CALL EXEC(20,0,IVT,IPRM(2),JDUM,IDUM,ICLAS)
      CALL EXEC(9,NAM,IPRM(1),IPRM(2),IPRM(3),ICLAS)
1005  IFLG=0
      ICLAS=0
      IPRM(1)=1
      IPRM(2)=ISN
      IPRM(3)=35
```

```
      IPRM(4)=IV
      CALL EXEC(20,0,LINE,35,JDUM,IDUM,ICLAS)
      CALL EXEC(9,NAM,IPRM(1),IPRM(2),IPRM(3),IPRM(4),ICLAS)
      GO TO 6
1010  IF(ITERM.EQ.1)GO TO 1000
      IF(ITERM.EQ.2)GO TO 7
      GO TO (611,611,450,611,400,903,907,961,611),KK
1020  IF(ITERM.EQ.2)GO TO 7
      GO TO 800
9999  END
      END$
```

```
*SKP1  T=00004 IS ON CR00002 USING 00002 BLKS R=0013

FTN4,L
      SUBROUTINE SKP1
C ROUTINE SKIPS CHARS UNTIL SPACE OCCURS THEN FINDS NEXT
C CHAR AFTER THE SPACE AND RETURNS WITH CORRECT PTR IN LOC J
      COMMON J,IARAY(72),IVT(90),ISN,IV,NBR1
      COMMON NBR2,NBR3,NBR01,IREL,ITERM,KV
      COMMON LINE(15),IAFLG(5),INDAY(15)
10    J=J+1
      IF(J.GT.72) GO TO 30
      IF(IARAY(J).EQ.40B) GO TO 20
      IF(IARAY(J).EQ.44B) GO TO 40
      GO TO 10
20    J=J-1
      CALL SKPSP
      RETURN
30    ITERM=2
      RETURN
40    ITERM=1
      RETURN
      END
      END$
```

```
*SKTNB T=00004 IS ON CR00002 USING 00002 BLKS R=0013

FTN4,L
      SUBROUTINE SKTNB
C ROUTINE SKIPS ALL CHARS UNTIL NUMBER IS REACHED THEN RETURNS
C WITH CORRECT PTR IN LOC J
      COMMON J,IARAY(72),IVT(90),ISN,IV,NBR1
      COMMON NBR2,NBR3,NBRO1,IREL,ITERM,KV
      COMMON LINE(15),IAFLG(5),INDAY(15)
10    CALL SKPSP
      IF(ITERM.EQ.1) RETURN
      IF(ITERM.EQ.2) RETURN
      DO 15 K=1,12
      IF(IARAY(J).EQ.(47+K)) GO TO 20
15    CONTINUE
      GO TO 10
20    RETURN
      END
      END$
```

*SKPSP T=00004 IS ON CR00002 USING 00002 BLKS R=0012

```
FTN4,L
      SUBROUTINE SKPSP
C ROUTINE SKIPS SPACES AND UPDATES PTR IN LOC J.  IF 72 CHARS
C ARE COUNTED THE FLAG ITERM IS SET TO 2.  IF A TERMINATING
C CHAR IS REACHED THE FLAG ITERM IS SET TO 1.
      COMMON J,IARAY(72),IVT(90),ISN,IV,NBR1
      COMMON NBR2,NBR3,NBRO1,IREL,ITERM,KV
      COMMON LINE(15),IAFLG(5),INDAY(15)
10    J=J+1
      IF(J.GT.72) GO TO 30
      IF(IARAY(J).EQ.44B) GO TO 40
      IF(IARAY(J).EQ.40B) GO TO 10
      RETURN
30    ITERM=2
      RETURN
40    ITERM=1
      RETURN
      END
      END$
```

```
*NBRTY T=00004 IS ON CR00002 USING 00006 BLKS R=0053

FTN4,L
      SUBROUTINE NBRTY
C ROUTINE SETS UP PROPER NUMBER TYPE,D1,D2 IN WDS NBR1,NBR2,NBR3
      COMMON J,IARAY(72),IVT(90),ISN,IV,NBR1
      COMMON NBR2 NBR3,NBR01,IREL,ITERM,KV
      COMMON LINE(15),IAFLG(5),INDAY(15)
10    DO 15 K=1,10
      IF(IARAY(J).EQ.(57B+K)) GO TO 50
15    CONTINUE
C A(V) ?
      IF(IARAY(J+1).EQ.50B) GO TO 60
      IF(IARAY(J+1).EQ.52B) GO TO 70
C VARIABLE LOGIC
      NBR1=2
      NBR3=0
      NBR2=IARAY(J)-100B
      CALL SKPSP
      RETURN
70    DO 75 I=1,5
      IF(IARAY(J).EQ.IAFLG(I))GO TO 80
75    CONTINUE
      ITERM=1
      RETURN
80    I=I*3
      NBR1=5
      NBR2=INDAY(I)
      NBR3=INDAY(I-1)
      CALL SKP1
      RETURN
50    NBR1=1
      NBR3=0
      NBR2=IARAY(J)-60B
51    CALL SKPSP
      IF(ITERM.EQ.1) RETURN
      IF(ITERM.EQ.2) RETURN
      DO 52 K=1,10
      IF(IARAY(J).EQ.(47+K)) GO TO 54
52    CONTINUE
      RETURN
54    NBR2=NBR2*10+(IARAY(J)-60B)
      GO TO 51
C A(V) AND A(C) LOGIC
60    DO 90 I=1,5
      IF(IARAY(J).EQ.IAFLG(I))GO TO 92
90    CONTINUE
      ITERM=1
      RETURN
92    I=I*3
      NBR2=INDAY(I)
      J=J+2
      DO 62 K=1,10
      IF(IARAY(J).EQ.(47+K)) GO TO 64
62    CONTINUE
```

```
      NBR1=4
      NBR3=IARAY(J)-100B
      CALL SKPSP
      IF(ITERM.EQ.1) RETURN
      IF(ITERM.EQ.2) RETURN
      IF(IARAY(J).NE.51B)ITERM=1
      CALL SKPSP
      RETURN
64    NBR1=3
      NBR3=IARAY(J)-60B
65    CALL SKPSP
      IF(ITERM.EQ.1) RETURN
      IF(ITERM.EQ.2) RETURN
      DO 66 K=1,10
      IF(IARAY(J).EQ.(47+K)) GO TO 69
66    CONTINUE
      IF(IARAY(J).NE.51B) ITERM=1
      CALL SKPSP
      RETURN
69    NBR3=NBR3*10+(IARAY(J)-60B)
      GO TO 65
      END
      END$
```

```
*NBRO  T=00003 IS ON CR00002 USING 00024 BLKS R=0000

FTN4,L
      SUBROUTINE NBRO
C CONVERTS AN ASCII NUMBER TO AN OCTAL VALUE
      COMMON J,IARAY(72),IVT(90),ISN,IV,NBR1
      COMMON NBR2,NBR3,NBRO1,IREL,ITERM,KV
      COMMON LINE(15),IAFLG(5),INDAY(15)
      NBRO1=0
10    DO 15 K=1,8
      IF(IARAY(J).EQ.(47+K)) GO TO 20
15    CONTINUE
      RETURN
20    NBRO1=NBRO1*8+(IARAY(J)-60B)
      CALL SKPSP
      IF (ITERM.EQ.1) RETURN
      IF (ITERM.EQ.2) RETURN
      GO TO 10
      END
      END$
```

```
*SVARA T=00004 IS ON CR00002 USING 00002 BLKS R=0011

FTN4,L
      SUBROUTINE SVARA
C  ROUTINE SETS UP 3 WDS IN VAR TABLE (IVT) EQUAL TO VALUES
C  IN NBR1,NBR2, AND NBR3.
      COMMON J,IARAY(72),IVT(90),ISN,IV,NBR1
      COMMON NBR2,NBR3,NBR01,IREL,ITERM,KV
      COMMON LINE(15),IAFLG(5),INDAY(15)
      IVT(KV)=NBR1
      KV=KV+1
      IVT(KV)=NBR2
      KV=KV+1
      IVT(KV)=NBR3
      KV=KV+1
      RETURN
      END
      END$
```

```
*SDS   T=00003 IS ON CR00002 USING 00004 BLKS R=0000

FTN4,L
      PROGRAM SDS
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISYST,IBUF(180)
      DIMENSION NAM(3),NAM1(3),IPRAM(5)
      DATA NAM/2HT2,2HXX,2HX /
      DATA NAM1/2HT1,2HXX,2HX /
      CALL RMPAR(IPRAM)
      IF(IPRAM(1).LT.100)GO TO 3
      IPRAM(1)=IPRAM(1)-100
      GO TO 56
3     DO 1 I=1,15
      DO 1 J=1,30
1     ISTAB(I,J)=0
      DO 2 I=1,4
      DO 2 J=1,30
2     ISTFL(I,J)=0
      DO 5 I=1,3
      DO 5 J=1,30
5     MRESP(I,J)=0
      DO 15 I=1,7
      DO 15 J=1,20
15    ISCB(I,J)=0
      DO 20 I=1,2
      DO 20 J=1,226
20    IVAR(I,J)=0
      DO 25 I=1,12
25    IRESP(I)=0
      DO 30 I=1,15
30    INDAY(I)=0
      DO 35 I=1,200
35    IVT(I)=0
      DO 40 I=1,5
40    IAFLG(I)=0
      ITOP=0
      INXT=1
      TR=0.0
      INDXR=0
      JJ=1
      DO 45 I=1,180
45    IBUF(I)=0
      DO 50 I=1,30
50    TREQ(I)=0.0
      DO 55 I=1,2
      DO 55 J=1,30
55    ITREQ(I,J)=0
      CALL NTSK1
      CALL NTSK2
      CALL NTSK3
      CALL NTSK4
```

```
      CALL NTSK5
      CALL NTSK6
      CALL NTSK7
      CALL EXEC(6,0,1)
56    CALL RMPAR(IPRAM)
      CALL MPNRM
      DO 59 I=1,12
      CALL EVSNS(1,I-1,0,NAM1,IERR)
      IF(IERR.EQ.1)GO TO 59
59    CONTINUE
      CALL EXEC(9,NAM,IPRAM(1),IPRAM(2),IPRAM(3),IPRAM(4),IPRAM(5))
      CALL EXEC(7)
      CALL MPNRM
      END
      END$
```

```
*EXPR  T=00003 IS ON CR00002 USING 00024 BLKS R=0000

FTN4
      SUBRUUTINE EXPR
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION ISTIM(9),IPRAM(5),IVFLG(5),NAM(3)
      DATA IP/2/
      DATA NAM/2HT3,2HXX,2HX /
C GET PARMS
      CALL RMPAR(IPRAM)
      DO 10 I=1,5
10    IPRM(I)=IPRAM(I)
C IPRM(1) = 1 START EVENT LOGIC
C           2 RESPONSE EVENT LOGIC
C           3 TIME EVENT LOGIC
C
C
C ILOG(1) = ID NUMBER
C     (2) = EXP NUMBER
C     (3) = STATE NUMBER
C     (4) = RESP BIT NUMBER
C     (5) THRU (10) = TIME AND YEAR
C
      GO TO (100,200,300),IPRM(1)
C START EVENT LOGIC
100   ISTSW=1
      ILOG(2)=IPRM(2)
      CALL EXEC(11,ILOG(5),ILOG(10))
      ILOG(1)=1000B
C LOG START OF EXPERIMENT
      ILOG(3)=0
      ILOG(4)=0
      IP=2
      CALL LOGDA
      ISSFL=0
      ILAST=0
      ISUPER=0
      NS=IPRM(4)
105   IFLG=0
      IST=NS
      IF(ISTAB(14,IST).GT.0)CALL INITL
      IF(ISSFL.GT.0)GO TO 110
      CALL INSCB(IND)
110   ISUPER=ILAST
      IF(ISUPER.LE.0)GO TO 111
      IF(ISCB(4,ISUPER).NE.IND)ISCB(4,ISUPER)=IND
111   ILAST=IND
      IF(ISTAB(13,IST).EQ.0)GO TO 120
      CALL INSCB(INDSS)
```

```
C SET UP COUNT OPERAND AND NEXT STATE
120   CALL EVVAR(ISTAB(10,IST),NS,0)
      ISCB(1,IND)=NS
      LOGAO=0
      CALL EVVAR(ISTAB(4,IST),ICT,0)
      CALL EVVAR(ISTAB(7,IST),IOP,0)
C RELATIONAL TRANSACTION ?
      IF(ISTAB(1,IST).GT.3)GO TO 197
C IF, IF RB, OR AFTER TRANSACTION ?
      IF(ISTAB(1,IST).NE.3)GO TO 125
C FOLLOWING TRANSACTION - SET UP FOL TABLE FROM OPERAND
      ISTFL(4,IOP)=IND
      GO TO 145
C IF OR AFTER TRANSACTION ?
125   IF(ISTAB(1,IST).GE.0)GO TO 130
C IF RB TRANSACTION - SET UP RESP AND MRESP TABLES
      MRESP(3,IST)=IND
      MRESP(1,IST)=IOP
      MRESP(2,IST)=IOP
      J=1
      DO 126 I=1,12
      ITES=IAND(IOP,J)
      IF(ITES.GT.0)IRESP(I)=-IST
126   J=ISHFT(J,1)
      GO TO 145
C AFTER TRANSACTION ?
130   IF(ISTAB(1,IST).NE.1)GO TO 140
C IF TRANSACTION - SET UP RESP TABLE
      IRESP(IOP)=IND
      ISCB(7,IND)=IOP
      GO TO 145
140   IF(ISTAB(1,IST).NE.2)GO TO 500
C AFTER TRANSACTION - SET UP COUNTER MAKE TIME REQUEST
      CALL SCHED(ICT,IND,INDXR)
      ISCB(6,IND)=INDXR
      GO TO 150
145   ISCB(6,IND)=ICT
150   ISCB(2,IND)=IST
      ISCB(3,IND)=ISUPER
      ISCB(4,IND)=INDSS
C LOGICAL FUNCTIONS ?
      IF(IFLG.EQ.1)GO TO 152
      IF(ISTAB(2,IST).EQ.0)GO TO 165
      IFLG=1
152   IF(ISTAB(2,IST).NE.0)GO TO 160
C LOGICAL FUNCTION - SET UP AND/OR PTR AS + OR -AND
      ILAS=ILAST
      IF(ISTAB(2,ISVST).EQ.1)ILAS=-ILAST
      IFLG=0
      ISCB(5,IND)=ILAS
      IST=ISCB(2,ILAST)
      GO TO 165
C GET INDEX FOR AND/OR SCB
160   CALL INSCB(INDX)
      INDAO=INDX
      IF(ISTAB(2,IST).EQ.1)INDAO=-INDX
      ISCB(5,IND)=INDAO
```

```
      IND=INDX
C MOVE MINOR STATE FROM VARIABLE TABLE
      ITEMP=30
161   IF(ISTAB(1,ITEMP).EQ.0)GO TO 164
      ITEMP=ITEMP-1
      GO TO 161
164   DO 162 I=1,15
162   ISTAB(I,ITEMP)=ISTAB(I,IST)
      IPTR=ISTAB(3,ITEMP)
      DO 163 I=1,9
163   ISTAB(I,ITEMP)=IVT(IPTR-1+I)
      ISVST=IST
      IST=ITEMP
      LOGAO=1
C LOG START OF STATE
165   ILOG(1)=2000B
      CALL EXEC(11,ILOG(5),ILOG(10))
      ILOG(3)=IST
      ILOG(4)=0
      CALL LOGDA
      IF(LOGAO.EQ.1)GO TO 120
      K=1
C STIMULUS ?
      I=2
      CALL EVVAR(ISTFL(1,IST),IWD,0)
      IF(IWD.EQ.0)GO TO 170
      CALL ONOFF(K,IWD,I)
C STATE HAVE A SUBSTATE ?
170   IF(ISTAB(13,IST).EQ.0)GO TO 185
      IND=INDSS
      INDSS=0
      ISSFL=1
      NS=ISTAB(13,IST)
      GO TO 105
185   ISSFL=0
C START OF EXPERIMENT FLAG SET ?
      IF(ISTSW.EQ.1)GO TO 195
      GO TO 400
195   ISTSW=0
      GO TO 400
C RELATIONAL TRANSACTION - SET UP COUNT,OPERAND,AND PTR SCB
197   CALL EVVAR(ISTAB(4,IST),INDX1,-1)
      CALL EVVAR(ISTAB(7,IST),INDX2,-1)
      ISCB(6,IND)=INDX1
      ISCB(7,IND)=INDX2
      IVAR(2,INDX1)=IND
      IVAR(2,INDX2)=IND
      JF=0
      IVFLG(1)=10
      DO 198 IF=2,5
      IF(IVFLG(IF).EQ.0)JF=1
      IF(IVFLG(IF).EQ.0)IVFLG(IF)=IND
      IF(JF.EQ.1)GO TO 199
198   CONTINUE
199   GO TO 150
C RESPONSE EVENT LOGIC
200   ILOG(4)=IPRM(2)
```

```
      IF(IRESP(ILOG(4)).LT.0)GO TO 210
      IND=IRESP(ILOG(4))
      ILOG(1)=40008
C LOG RESPONSE EVENT
      ILOG(3)=ISCB(2,IND)
      CALL LOGDA
C INCORRECT RESPONSE ?
      IF(ISCB(7,IND).NE.ILOG(4))GO TO 500
C DECREMENT COUNT - COUNT COMPLETE ?
205   ISCB(6,IND)=ISCB(6,IND)-1
      IF(ISCB(6,IND).NE.0)GO TO 500
      IFLAG=1
C SET UP TO EXIT
      GO TO 215
210   INDX=-IRESP(ILOG(4))
      IBITP=MRESP(1,INDX)
      IBCK=IAND(ILOG(4),IBITP)
      IF(IBCK.EQ.0)GO TO 212
      MRESP(1,INDX)=IEOR(IBITP,ILOG(4))
212   ILOG(1)=40008
      ILOG(3)=ISCB(2,IND)
      CALL LOGDA
C LOG MULTIPLE RESPONSE EVENT
      IND=MRESP(3,INDX)
      IF(MRESP(1,INDX).NE.0)GO TO 500
C RESET MULTIPLE RESPONSE
      MRESP(1,INDX)=MRESP(2,INDX)
      GO TO 205
C EXIT LOGIC
215   IF(ISCB(5,IND).GE.0)GO TO 231
      ISSFG=1
      I3=ISCB(3,IND)
      I4=ISCB(4,IND)
      I5=-ISCB(5,IND)
      IF(I3.EQ.0)GO TO 246
      IF(ISCB(4,I3).EQ.IND)ISCB(4,I3)=I5
246   IF(I4.EQ.0)GO TO 247
      IF(ISCB(3,I4).EQ.IND)ISCB(3,I4)=I5
247   IF(ISCB(5,I5).NE.-IND)GO TO 248
      ISCB(5,I5)=0
      GO TO 249
248   IPTR=I5
241   IF(-ISCB(5,IPTR).LT.IPTR)GO TO 242
      IPTR=-ISCB(5,IPTR)
      GO TO 241
242   IF(-ISCB(5,IPTR).EQ.IND)ISCB(5,IPTR)=ISCB(5,IND)
249   IF(ISTAB(1,ISCB(2,IND)).EQ.3)GO TO 232
      GO TO 220
231   ISSFG=0
      NS=0
232   ISYSS=ISCB(4,IND)
220   GO TO (221,222,223,216,216,224),IFLAG
C EXIT RESPONSE
221   IF(IRESP(ILOG(4)).LT.0)GO TO 208
      IRESP(ILOG(4))=0
      GO TO 224
208   IS=-IRESP(ILOG(4))
      DO 209 I=1,3
```

```
 209    MRESP(I,IS)=0
        IRESP(ILOG(4))=0
        GO TO 224
C EXIT FOLLOWING
 222    IST=ISCB(2,IND)
        ISTFL(4,ISVST)=0
        GO TO 224
C EXIT AFTER
 223    ITREQ(2,ISCB(6,IND))=-1
        ISCB(6,IND)=0
        GO TO 224
 216    IF(IFLAG.EQ.5)GO TO 224
        IVAR(ISCB(6,IND),2)=0
 224    ISVST=ISCB(2,IND)
        ISVAO=ISCB(5,IND)
        IF(ISSFG.NE.0)GO TO 225
        NS=ISCB(1,IND)
 225    IS4=ISTFL(4,ISVST)
        IF(IS4.EQ.0)GO TO 230
        ISCB(6,IS4)=ISCB(6,IS4)-1
        IF(ISCB(6,IS4).NE.0)GO TO 230
        IND=IS4
        IFLG=6
        IFLAG=2
        GO TO 215
 230    ISSFG=1
        K=0
        I=2
        CALL EVVAR(ISTFL(1,ISVST),IWD,0)
        IF(IWD.EQ.0)GO TO 235
        CALL ONOFF(K,IWD,I)
 235    IF(IFLG.EQ.7)GO TO 236
        ILAST=ISCB(3,IND)
        IF(IFLG.EQ.6)IFLG=7
 236    ISCB(2,IND)=0
        ISCB(5,IND)=0
        ISCB(7,IND)=0
        IF(ISVST.GE.25)ISTAB(1,ISVST)=0
        CALL EXEC(11,ILOG(5),ILOG(10))
        ILOG(1)=3000B
C LOG END OF STATE
        ILOG(3)=ISVST
        ILOG(4)=0
        CALL LOGDA
C AND/OR STATE TO CLEAR ?
        IF(ISVAO.LE.0)GO TO 240
        IAOFG=1
        IND=ISVAO
        GO TO 255
C SUBSTATE TO CLEAR ?
 240    IF(ISVSS.EQ.0)GO TO 245
        IF(ISCB(2,ISVSS).EQ.0 .AND. IFLG.EQ.3)GO TO 245
        IND=ISVSS
        GO TO 255
C NEXT STATE TO INSTALL ?
```

```
245    IF(NS.NE.0)GO TO 105
       IF(ISCB(3,IND).NE.0)IFLG=3
       ISS=ISCB(3,IND)
       IF(ISCB(2,ISS).NE.0)GO TO 400
C END OF EXPERIMENT ?
       CALL EXEC(11,ILOG(5),ILOG(10))
       CALL EXEC(12,NAM,4,1,-100)
       WRITE(1,250)ILOG(2)
250    FORMAT("END OF EXP.",I6,/)
       ILOG(1)=50008
C LOG END OF EXPERIMENT
       ILOG(3)=0
       ILOG(4)=0
       CALL LOGDA
       GO TO 500
C SUBSTATE CLEAR LOGIC
255    IST=ISCB(2,IND)
       IF(IST.NE.0)GO TO 252
       IF(IAOFG.EQ.1)GO TO 251
       GO TO 245
251    IAOFG=0
       GO TO 240
C RELATIONAL TRANSACTION ?
252    IF(ISTAB(1,IST).LE.3)GO TO 260
       IF(ISCB(7,IND).LT.0)GO TO 256
       IVAR(ISCB(7,IND),2)=0
256    IFLAG=4
       GO TO 220
260    IF(ISTAB(1,IST).LT.0)GO TO 263
       GO TO (261,262,220),ISTAB(1,IST)
C RESPONSE EVENT
261    CALL EVVAR(ISTAB(7,IST),ILOG(4),0)
       IFLAG=1
       GO TO 220
C AFTER EVENT
262    IFLAG=3
       GO TO 220
C MULTIPLE RESPONSE
263    IBP=ISTAB(IST,8)
       IMASK=1
       DO 264 I=1,12
       ICK=IAND(IBP,IMASK)
       IF(ICK.EQ.1)IMR=IRESP(I)
       IF(ICK.EQ.1)IRESP(I)=0
264    IBP=ISHFT(IBP,-1)
       MRESP(IMR,3)=0
       IFLAG=6
       GO TO 220
C TIME EVENT LOGIC
300    CALL EXEC(11,ILOG(5),ILOG(10))
       IND=IPRM(2)
       IFLAG=3
       GO TO 215
C RELATIONAL EVENT LOGIC
400    IF(IVFLG(1).NE.10)GO TO 500
401    IF(IVFLG(IP).EQ.0)GO TO 402
```

```
      IP=IP+1
      INDX=IVFLG(IP-1)
      IST=ISCB(2,INDX)
      ITYPE=ISTAB(1,IST)-3
  405 IF(ISCB(7,INDX).LT.0)GO TO 406
      IVB=IVAR(1,ISCB(7,INDX))
      GO TO 407
  402 IF(IP.GE.5)GO TO 403
      IP=IP+1
      GO TO 401
  403 IP=2
      GO TO 500
  406 IVB=-ISCB(7,INDX)
  407 IVA=IVAR(1,ISCB(6,INDX))
      GO TO(410,415,420,425,430,435),ITYPE
  410 IF(IVA.EQ.IVB)GO TO 450
      GO TO 475
  415 IF(IVA.LT.IVB)GO TO 450
      GO TO 475
  420 IF(IVA.GT.IVB)GO TO 450
      GO TO 475
  425 IF(IVA.LE.IVB)GO TO 450
      GO TO 475
  430 IF(IVA.GE.IVB)GO TO 450
      GO TO 475
  435 IF(IVA.NE.IVB)GO TO 450
      GO TO 475
  450 IF(ISCB(6,INDX).GT.0)IVAR(2,ISCB(6,INDX))=0
      IF(ISCB(7,INDX).GT.0)IVAR(2,ISCB(7,INDX))=0
      IND=INDX
      IFLAG=6
      GO TO 215
  475 IF(IP.LT.6)GO TO 400
      IF(IP.GE.6)IP=2
  500 CALL EXEC(6,0,2)
      RETURN
      END
      END$
```

*ONOFF T=00003 IS ON CR00002 USING 00002 BLKS R=0000

```
FTN4,L
      SUBROUTINE ONOFF(K,IWD,J)
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCD(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDRY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION IWD(1)
      INUM=1
      ICHAN=4
      IF(J.GE.2)ICHAN=2
      IF(K.EQ.0)GO TO 100
      CALL DOL(INUM,ICHAN,IWD,IWD,IERR)
      GO TO 200
100   ICOMP=IEOR(177777B,IWD)
      CALL DOL(INUM,ICHAN,ICOMP,IWD,IERR)
200   RETURN
      END
      END$
```

```
*INSCB T=00003 IS ON CR00002 USING 00001 BLKS R=0000

FTN4,L
      SUBROUTINE INSCB(IRTN)
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOF,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DO 100 I=1,20
      IF(ISCB(2,I).EQ.0)IRTN=I
      IF(ISCB(2,I).EQ 0)GO TO 200
100   CONTINUE
200   ISCB(2,I)=1
      RETURN
      END
      END$
```

```
*DOOPS T=00003 IS ON CR00002 USING 00024 BLKS R=0000

FTN4,L
      SUBROUTINE DOOPS
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION IREC(90),IPRAM(5)
      CALL RMPAR(IPRAM)
      IF(IPRAM(1).EQ.3)GO TO 50
      IF(IPRAM(1).EQ.50)GO TO 60
      CALL EXEC(21,IPRAM(5),IREC,35)
      DO 30 I=1,15
30    ISTAB(I,IPRAM(2))=IREC(I)
      DO 31 I=16,20
31    IAFLG(I-15)=IREC(I)
      DO 32 I=21,35
32    INDAY(I-20)=IREC(I)
      GO TO 80
50    DO 55 I=1,3
55    ISTFL(I,IPRAM(5))=IPRAM(I+1)
      GO TO 80
60    CALL EXEC(21,IPRAM(4),IREC,IPRAM(2))
      DO 70 I=1,IPRAM(2)
70    IVT(IPRAM(3)+(I-1))=IREC(I)
80    CALL EXEC(6)
      RETURN
      END
      END$
```

```
*EVVAR T=00003 IS ON CR00002 USING 00003 BLKS R=0000

FTN4,L
      SUBROUTINE EVVAR(IADR,IRTN,IFLG)
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCHT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION IADR(3)
      IF(IADR(1).EQ.0)IRTN=0
      IF(IADR(1).EQ.0)GO TO 500
      IF(IFLG.LT.0)GO TO 90
      GO TO (1,2,3,4),IADR(1)
C GET CONSTANT VALUE FROM WORD 2
1     IRTN=IADR(2)
      GO TO 500
C USING PTR IN WORD 2 GET VALUE FROM VARIABLE TABLE
2     IRTN=IVAR(1,IADR(2))
      GO TO 500
C USING PTRS IN WORDS 2 AND 3 GET VALUE FROM VARIABLE TABLE
3     INDX=IADR(2)-1+IADR(3)
      IRTN=IVAR(1,INDX)
      GO TO 500
4     INDX=IVAR(IADR(3))+IADR(2)-1
      IRTN=IVAR(1,INDX)
      GO TO 500
90    GO TO (100,100,300,400),IADR(1)
100   IRTN=IADR(2)
      GO TO 500
300   IRTN=IADR(2)+IADR(3)-1
      GO TO 500
400   INDX=IVAR(1,IADR(3))
      IRTN=IADR(2)+INDX-1
500   RETURN
      END
      END$
```

```
*INITL T=00003 IS ON CR00002 USING 00003 BLKS R=0000

FTN4,L
      SUBROUTINE INITL
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION IPARM(5),NAM(3)
      DATA NAM/2HT7,2HXX,2HX /
      ISUM=0
      IF(ISTAB(14,IST).NE.1)GO TO 60
      INDX=ISTAB(15,IST)
40    CALL EVVAR(IVT(INDX+3),IEQ,-1)
      ILGTH=IVT(INDX)
      CALL EVVAR(IVT(INDX+6),IRTN,0)
      ISUM=ISUM+IRTN
      IF(ILGTH.NE.12)GO TO 50
      CALL EVVAR(IVT(INDX+9),IRTN,0)
      ISUM=ISUM+IRTN
50    IVAR(1,IEQ)=ISUM
      IF(IVT(INDX+2).EQ.0)GO TO 500
      INDX=IVT(INDX+2)
      ISUM=0
      GO TO 40
60    IF(ISTAB(14,IST).LT.2)GO TO 500
      IF(ISTAB(14,IST).GT.5)GO TO 500
      IPARM(1)=ISTAB(14,IST)-1
      IPARM(2)=ISTAB(15,IST)
      IPARM(3)=0
      IF(IVT(ISTAB(15,IST)+3).EQ.5)IPARM(3)=1
      CALL EXEC(24,NAM,IPARM(1),IPARM(2),IPARM(3))
500   RETURN
      END
      END$
```

```
*RESP  T=00003 IS ON CR00002 USING 00002 BLKS R=0000

FTN4,L
      SUBROUTINE RESP
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISYST,IBUF(180)
      DIMENSION NAM(3),IPRAM(5)
      DATA NAM/2HT2,2HXX,2HX /
      CALL RMPAR(IPRAM)
      CALL EXEC(11,ILOG(5),ILOG(10))
      IPRAM(2)=IPRAM(3)+1
      IPRAM(1)=2
      CALL EXEC(24,NAM,IPRAM(1),IPRAM(2))
      CALL EXEC(6)
      RETURN
      END
      END$
```

```
*LOGG  T=00003 IS ON CR00002 USING 00002 BLKS R=0000

FTN4,L
      SUBROUTINE LOGG
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION IPRAM(5)
      CALL RMPAR(IPRAM)
C IPRAM(1) = STARTING ADDRESS
C IPRAM(2) = LENGTH
C IPRAM(3) = 5 IF END OF EXPERIMENT
      CALL EXEC(2,10B,IBUF(IPRAM(1)),IPRAM(2))
      IF(IPRAM(3).NE.5)GO TO 10
C WRITE END OF FILE AND REWIND TAPE
      CALL EXEC(3,110B)
      CALL EXEC(3,410B)
10    CALL EXEC(6,0,2)
      RETURN
      END
      END$
```

```
*LOGDA T=00003 IS ON CR00002 USING 00006 BLKS R=0000

FTN4,L
      SUBROUTINE LOGDA
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION NAM(3)
      DATA NAM/2HT5,2HXX,2HX /
      ICK=ILOG(1)/1000B
      K=1
      DO 5 I=JJ,JJ+8
      IBUF(I)=ILOG(K)
 5    K=K+1
      JJ=JJ+9
      IF(JJ.EQ.181)GO TO 10
      IF(JJ.EQ.91)GO TO 20
      GO TO 30
 10   IPRM(1)=91
      IPRM(2)=90
      IPRM(3)=ICK
      IF(ICK.EQ.5)GO TO 40
      JJ=1
      GO TO 45
 20   IPRM(1)=1
      IPRM(2)=90
      IPRM(3)=ICK
      IF(ICK.EQ.5)GO TO 31
      GO TO 45
 30   IF(ICK.NE.5)GO TO 50
      IF(JJ.GT.91)GO TO 40
 31   IPRM(1)=1
      IPRM(2)=JJ-1
      IPRM(3)=5
      GO TO 45
 40   IPRM(1)=91
      IPRM(2)=JJ-91
      IPRM(3)=5
 45   CALL EXEC(24,NAM,IPRM(1),IPRM(2),IPRM(3))
 50   RETURN
      END
      END$
```

```
*RDWRT T=00004 IS ON CR00002 USING 00012 BLKS R=0084

FTN4,L
          SUBROUTINE RDWRT
          COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
          COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
          COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
          COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
          COMMON INDXR,IPRM(5),IST,JJ
          COMMON ISVST,IBUF(180)
          DIMENSION NAM1(3),IDCB1(144),LINEA(36)
          DIMENSION IDCB(144),NAM(3),LINE(36),IPRAM(5)
          DATA NAM/2HOP,2HIN,2H  /
          DATA NAM1/2HOP,2HOU,2HT /,IOP/0/,IOPN/0/
          CALL RMPAR(IPRAM)
          IF(IPRAM(1).EQ.1)GO TO 30
5         IF(IOP.EQ.1)GO TO 10
          IOP=1
          CALL OPEN(IDCB,IERR,NAM)
10        ICLAS=0
          CALL READF(IDCB,IERR,LINE,36)
          IF(LINE(1).EQ.2HEN)IOP=0
          IF(LINE(1).EQ.2HEN)CALL CLOSE(IDCB,IERR)
          CALL EXEC(20,0,LINE,36,JDUM,IDUM,ICLAS)
          IPRAM(2)=ICLAS
          CALL PRTN(IPRAM)
50        IF(IERR.LT.0)WRITE(1,100)IERR
100       FORMAT("RDWRT ERROR",I6)
          CALL EXEC(6,0,1)
          CALL RMPAR(IPRAM)
          IF(IPRAM(1).EQ.1)GO TO 30
          IF(IPRAM(1).EQ.0)GO TO 5
          IF(IPRAM(1).EQ.3)GO TO 20
          GO TO 115
20        CALL OPEN(IDCB,IERR,NAM)
          CALL OPEN(IDCB1,IERR,NAM1)
21        CALL READF(IDCB1,IERR,LINE,36)
          IF(IERR.EQ.-12)IEND=1
          IF(IERR.EQ.-12)GO TO 23
          CALL WRITF(IDCB,IERR,LINE,36)
          IF(IEND.NE.1)GO TO 21
23        CALL CLOSE(IDCB1,IERR)
          CALL CLOSE(IDCB,IERR)
          GO TO 115
30        IF(IOPN.EQ.1)GO TO 40
          IOPN=1
          IF(IPRAM(2).EQ.1)GO TO 35
          CALL OPEN(IDCB1,IERR,NAM1)
          GO TO 40
35        CALL OPEN(IDCB1,IERR,NAM1)
37        CALL POSNT(IDCB1,IERR,1)
          IF(IERR.EQ.0)GO TO 37
          CALL POSNT(IDCB1,IERR,-2)
40        CALL EXEC(21,IPRAM(3),LINEA,36)
          CALL WRITF(IDCB1,IERR,LINEA,36)
          IF(LINEA(1).EQ.2HEN)IOPN=0
          IF(LINEA(1).EQ.2HEN)CALL CLOSE(IDCB1,IERR)
          GO TO 50
115       IF(IERR.LT.0)WRITE(1,100)IERR
          CALL EXEC(6)
          END
          END$
```

```
*TSCHD T=00003 IS ON CR00002 USING 00003 BLKS R=0000

FTN4,L
C THIS SUBROUTINE WILL BE USED TO SCHEDULE TASK EXPR WHEN
C DESIRED TIME HAS ELASPSED. TASK EXPR WILL BE INSTALLED
C IN CORE BY NTASK AS T2XXX.
      SUBROUTINE TSCHD
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION NAM(3)
      DATA NAM/2HT2,2HXX,2HX /
C PICK UP PARAMETER FROM TOP REQUEST.
10    IPRM(1)=3
      IPRM(2)=ITREQ(2,ITOP)
C GO SCHEDULDE TASK EXPR
      IF(ITREQ(2,ITOP).EQ.-1)GO TO 12
      CALL EXEC(24,NAM,IPRM(1),IPRM(2))
C DELETE TOP REQUEST
12    ITREQ(2,ITOP)=0
C SET UP TOP REQUEST TO POINT TO NEXT REQUEST
      INXT=ITOP
      IF(ITREQ(1,ITOP).EQ.0)GO TO 25
      ITOP=ITREQ(1,ITOP)
C TIME REQUEST OF 0 OFFSET GO BACK AND SKED EXPR WITH NEW PARMS
      IF(TREQ(ITOP).EQ.0.0)GO TO 10
      IF(TREQ(ITOP).LT.100.)GO TO 15
C SET UP PROPER TIME FOR EXEC CALL
      J=2
      ITR=TREQ(ITOP)/100.
      GO TO 20
15    J=1
      ITR=TREQ(ITOP)
C SCHEDULE TSCHD AFTER ITR ELASPSED TIME
20    CALL EXEC(12,0,J,0,-ITR)
      GO TO 10
25    ITOP=0
      CALL EXEC(6)
      RETURN
      END
      END$
```

```
*DELT  T=00003 IS ON CR00002 USING 00003 BLKS R=0000

FTN4,L
C SUBROUTINE DELT IS USED TO CALCULATE THE DIFFERANCE IN
C TWO TIMES. THE STARTING TIME IS LOCATED IN LOCATIONS
C ITTOP(1) THRU ITTOP(5) AND THE ENDING TIME IS LOCATED IN
C ITIME(1) THRU ITIME(5). THE CALCULATED DIFFERANCE IS
C RETURNED IN LOCATION DELTA AND IS IN 10'S OF MS.
      SUBROUTINE DELT(DELTA)
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION IBASE(5),IDELT(5)
      DATA IBASE/100,60,60,24,1/
C CALCULATE DIF IN ITTP1 AND ITIME
      DO 10 J=1,4
      IDELT(J)=ILOG(J+4)-ITTOP(J)
      IF(IDELT(J).GE.0)GO TO 10
      ILOG(J+5)=ILOG(J+5)-1
      ILOG(J+4)=ILOG(J+4)+IBASE(J)
      IDELT(J)=ILOG(J+4)-ITTOP(J)
10    CONTINUE
C CONVERT TO 10 OF MS
      DELTA=IDELT(1)+(IDELT(2)*100.)
      DELTA=DELTA+(IDELT(3)*6000.)
      DELTA=DELTA+(IDELT(4)*360000.)
100   RETURN
      END
      END$
```

```
*SCHED T=00003 IS ON CR00002 USING 00005 BLKS R=0000

FTN4,L
C SUBROUTINE SCHED IS USED TO THREAD IN TIME REQUESTS AS
C THEY OCCUR. THE INITIAL TIME REQUEST CAUSES TASK TSCHD
C TO BE SCHEDULED.
      SUBROUTINE SCHED(ISCHED,INDX,INDXP)
C VARIABLES USED IN THIS SUBROUTINE ARE DESCRIBED AS FOLLOWS:
C ISCHED- TIME REQUEST PASSED TO SCHED
C INDX- INDEX INTO SCB PASSED TO SCHED
C INDXR- INDEX INTO ITREQ AND TREQ PASSED TO CALLING PROG
      COMMON ISTAB(15,30),ISTFL(4,30),MRESP(3,30)
      COMMON IPCNT(4),ISCB(7,20),IND,IVAR(2,226)
      COMMON IRESP(12),IAFLG(5),INDAY(15),IVT(200),ILOG(10)
      COMMON TREQ(30),ITREQ(2,30),ITTOP(5),ITOP,INXT,TR
      COMMON INDXR,IPRM(5),IST,JJ
      COMMON ISVST,IBUF(180)
      DIMENSION NAM(3),ITTP(5)
      DATA NAM/2HT3,2HXX,2HX /
C TREQ- ARRAY OF TIME REQUESTS IN 10 OF MS
C ITREQ- PARALLELL ARRAY TO TREQ -INDEX TO TREQ AND SCB
C ITTOP- TIME OF TOP REQUEST
C ITOP- POINTER TO TOP REQUEST
C INXT- POINTER TO NEXT ENTRY IN ITREQ AND TREQ
C ITR- TIME REMAINING IN 10 OF MS OR IN SEC
C ITIME- ARRAY TO STORE CURRENT TIME IN
C IYEAR- CURRENT YEAR
C
C GET TIME REQ AND PTR TO SCB INTO PROPER ARRAYS
      IF(ISCHED.LT.0)GO TO 3
      TREQ(INXT)=ISCHED*100.0
      GO TO 5
3     TREQ(INXT)=-ISCHED
5     ITREQ(2,INXT)=INDX
C
C GET CURRENT TIME
      CALL EXEC(11,ILOG(5),ILOG(10))
C
C SEE IF THERE IS A TOP ENTRY
C
      IF(ITOP.NE.0)GO TO 20
C
C NO SO SET UP VARIABLES FOR A TOP REQ
C
      DO 10 I=1,5
10    ITTOP(I)=ILOG(I+4)
      ITOP=INXT
      TR=TREQ(INXT)
      TREQ(INXT)=0
      ITREQ(1,INXT)=0
      IF(TR.LT.100.)GO TO 12
      J=2
      ITR=TR/100.
      GO TO 15
12    J=1
      ITR=TR
```

```
C
C
C SKED TSCHD TO BE RUN AFTER OFFSET OF ITR
C
15      CALL EXEC(12,NAM,J,0,-ITR)
C
        GO TO 40
C
C YES THERE WAS A TOP ENTRY SO SEE IF THIS REQ IS LESS
C
20      CALL DELT(DELTA)
        TIME=TR-DELTA
        DIF=TIME-TREQ(INXT)
        ICK=ITOP
        IF(DIF.LT.0.0)GO TO 60
        TREQ(ICK)=DIF
        DO 30 I=1,5
30      ITTOP(I)=ILOG(I+4)
35      ITREQ(1,INXT)=ITOP
        ITOP=INXT
C
C SKED TSCHD TO BE RUN AFTER NEW TOP TIME
C
        TR=TREQ(ITOP)
        IF(TR.LT.100.)GO TO 25
        J=2
        ITR=TR/100.
        GO TO 26
25      J=1
        ITR=TR
26      CALL EXEC(12,NAM,J,0,-ITR)
C
C UPDATE PTR TO NEXT
C
40      IHDXP=INXT
        DO 45 J=1,30
        INUM=J
        IF(ITREQ(2,J).EQ.0)J=30
45      CONTINUE
50      INXT=INUM
        GO TO 100
60      TREQ(INXT)=-DIF
        IF(ITREQ(1,ICK).EQ.0)GO TO 70
        IPREV=ICK
        ICK=ITREQ(1,ICK)
        DIF=TREQ(ICK)-TREQ(INXT)
        IF(DIF.LT.0.0)GO TO 60
        TREQ(ICK)=DIF
        ITREQ(1,IPREV)=INXT
        ITREQ(1,INXT)=ICK
        GO TO 40
70      ITREQ(1,ICK)=INXT
        ITREQ(1,INXT)=0
        GO TO 40
100     RETURN
        END
        END$
```

APPENDIX C

SDS QUICK REFERENCE GUIDE

APPENDIX C

SDS QUICK REFERENCE GUIDE

APPENDIX C

SDS QUICK REFERENCE GUIDE

This appendix contains summary information about each of the SDS instructions.

1. JOB CONTROL

$ --used to terminate each source program line
NEW$--required first instruction in an SDS program
END$--required last instruction in an SDS program

2. AFTER transitional

AF K T
AF K S
AF K M
AF K H
AF V S
AF A(K) S
AF A(V) S

Where K = a constant value ranging from 1 to 32767;
V= any variable A-Y containing from 1 to 32767;
A(K) = any constant element of any array A-Z containing from 1 to 32767;
A(V) = any variable A-Y element of any array A-Z containing from 1 to 32767;
T = ticks of system clock in 10s of ms;
S = seconds;
M = minutes;
and H = hours.

3. FOLLOWING transitional

FO K S K
FO V S K
FO A(K) S K
FO A(V) S K
FO V S V
FO V S A(K)
FO V S A(V)
FO A(K) S V
FO A(K) S A(K)
FO A(K) S A(V)
FO A(V) S V
FO A(V) S A(K)
FO A(V) S A(V)

Where K = a constant value ranging from 1 to 32767 when defining a count;

K = a constant value ranging from 1 to 30 when defining a state number other than its own;

V = any variable A-Y containing from 1 to 32767 when defining a count;

V = any variable A-Y containing from 1 to 30 when defining a state number other than its own;

A (K) = any constant element of any array A-Z containing from 1 to 32767 when defining a count;

A (K) = any constant element of any array A-Z containing from 1 to 30 when defining a state number other than its own;

A (V) = any variable A-Y element of any array A-Z containing from 1 to 32767 when defining a count;

A (V) = any variable A-Y element of any array A-Z containing from 1 to 30 when defining a state number other than its own.

4. IF transitional

IF K R K
IF V R K
IF A(K) R K
IF A(V) R K
IF V R V
IF V R A(K)
IF V R A(V)
IF A(K) R V
IF A(K) R A(K)
IF A(K) R A(V)
IF A(V) R V
IF A(V) R A(K)
IF A(V) R A(V)

Where K = a constant value ranging from 1 to 32767 when defining a count;

K = a constant value ranging from 1 to 12 when defining a response bit number;

V = any variable A-Y containing from 1 to 32767 when defining a count;

V = any variable A-Y containing from 1 to 12 when defining a response bit number;

A (K) = any constant element of any array A-Z containing from 1 to 32767 when defining a count;

A (K) = any constant element of any array A-Z containing from 1 to 12 when defining a response bit number;

A (V) = any variable A-Y element of any array A-Z containing from 1 to 32767 when defining a count;

and A (V) = any variable A-Y element of any array A-Z containing from 1 to 12 when defining a response bit number.

5. IF transitional (binary)

IF K RB K
IF V RB K
IF A(K) RB K
IF A(V) RB K
IF V RB V
IF V RB A(K)
IF V RB A(V)
IF A(K) RB V
IF A(K) RB A(K)
IF A(K) RB A(V)
IF A(V) RB V
IF A(V) RB A(K)
IF A(V) RB A(V)

Where K=a constant value ranging from 1 to 32767 when defining a count;
K = a constant value ranging from 1 to 4095 when defining a multiple response pattern;
V = any variable A-Y containing from 1 to 32767 when defining a count;
V = any variable A-Y containing from 1 to 4095 when defining a multiple response pattern;
A(K) = any constant element of any array A-Z containing from 1 to 32767 when defining a count;
A(K) = any constant element of any array A-Z containing from 1 to 4095 when defining a multiple response pattern;
A(V) = any variable A-Y element of any array A-Z containing from 1 to 32767 when defining a count;
A(V) = any variable A-Y element of any array A-Z containing from 1 to 4095 when defining a multiple response pattern.

6. IF modified (relational)

IF V Z V
IF A(K) Z V
IF A(V) Z V
IF V X A(K)
IF V Z A(V)
IF A(K) Z A(K)
IF A(K) Z A(V)
IF A(V) Z A(K)
IF A(V) Z A(V)

Where V = any variable A-Y containing from -32768 to 32767;
A(K) = any constant element of any array A-Z containing from -32768 to 32767;
A(V) = any variable A-Y element of any array A-Z containing from -32768 to 32767;
and Z = any one of the relational operators EQ, NE, LT, GT, LE, or GE.

7. <u>STATE modifying or identifying</u>

ST K
STK

Where K = a constant value ranging from 1 to 30 except in programs using the logical instruction in which case the 30 would be reduced by one for each additional element of the logical instruction.

8. <u>THEN modifying or identifying</u>

TH K
TH V
TH A(K)
TH A(V)

Where K = a constant equal to the next desired state number;
V = any variable A-Y containing a value equal to the next desired state number;
A(K) = any constant element of any array A-Z containing a value equal to the next desired state number;
and A(V) = any variable A-Y element of any array A-Z containing a value equal to the next desired state number.

9. <u>VARIABLE modifying identifying</u>

VAR V=K
VAR V=V
VAR V=A(K)
VAR V=A(V)
VAR A(K)=K
VAR A(K)=V
VAR A(K)=A(K)
VAR A(K)=A(V)
VAR A(V)=K
VAR A(V)=V
VAR A(V)=A(K)
VAR A(V)=A(V)
VAR V=K, A(K)=V, A(V)=A(K) etc.
VAR X=X+K

Where K = a constant value ranging from -32768 to 32767;
V = any variable A-Y to be initialized;
V = any variable A-Y containing a value ranging from -32768 to 32767;
A(K) = any constant element of any array A-Z to be initialized;
A(K) = any constant element of any array A-Z containing a value ranging from -32768 to 32767;
A(V) = any variable A-Y element of any array A-Z to be initialized;
A(V) = any variable A-Y element of any array A-Z containing a value ranging from -32768 to 32767;
and X = either V, A(K), or A(V).

10. DIMENSION modifying or identifying

DIM A,L

Where A = any array name A-Z;
and L = the length of the array which cannot exceed 200 words. If more than one array is dimensioned (maximum of four) the combined total of the size of the arrays cannot exceed 200 words.

11. STIMULUS modifying or identifying

ST K
ST V
ST A(K)
ST A(V)
SB K
SB V
SB A(K)
SB A(V)

Where SB = an optional character set for the character set ST;
K = a constant value ranging from 1 to 4095;
V = any variable A-Y containing a value ranging from 1 to 4095;
A(K) = any constant element of array A-Z containing a value from 1 to 4095;
and A(V) = any variable A-Y element of any array A-Z containing a value ranging from 1 to 4095.

12. SUBSTATE modifying and identifying

SU K
SS K

Where   SS = an optional character set for the character set SU;
and   K = a constant value ranging from 1 to 30 and is equal to a state number that is defined in the SDS program.

13. AND/OR logical

X OR Y
X OR Y OR Z etc.
X AND Y
X AND Y AND Z etc.

Where X, Y, and Z = any of the transitional instructions.

NOTE: The use of LOGICAL instructions requires the use of a state table entry for each element X, Y, or Z and will therefore reduce the maximum number of states from 30 to 30 minus the number of additional logical instruction elements.

14. CRT, PTR, PUN, RDR input/output

CRT V
CRT A(K)
CRT A(V)
CRT V; A(K); A(V); V etc.
CRT A*

Where   V = any variable A-Y;
      A(K) = any constant element of any array A-Z;
      A(V) = any variable A-Y element of any array A-Z;
and   A* = any entire array A-Z.

NOTE: The instructions PTR, PUN, and RDR are written in the same format as the instruction CRT.

APPENDIX D

RTE-II INITIALIZATION PROCEDURE

APPENDIX D

RTE-II INITIALIZATION PROCEDURE

The procedure used to operate the HP-2100 computer using the RTE-II operating system is described in Table D-I. When the RTE-II system is initially brought up, it runs its file manager program FMGR which, in turn, runs a transfer file that produces the message in Figure D-1.

Table D-I

Initializing Procedure for RTE-II

1. Insert the disc cartridge containing the SDS system into the disk drive and move the disc load-unload switch to the load position. Wait for the "Drive Ready" light to illuminate.

2. Set P register to 77750_8.

3. Set S register to 0.

4. Depress External Preset, Internal Preset, and Run switches. The computer should halt with 102077_8 in the display register.

5. Depress the Run switch and the SDS welcome message should be printed on the computer console.

6. The SDS is now ready to be used as described in this report.

- - -

```
SET TIME
:SV,4
TE,*****
TE,***** WELCOME TO THE SDS PLEASE TYPE RU,OPCOM WHEN YOU
TE,***** ARE READY TO BEGIN USING THE SDS.
TE,*****
::
:
```

Figure D-1. The SDS Welcome Message

The last line in this welcome message contains a colon (:) which is the pr mpt character for the file manager program FMGR. The prompt character (:) indicates that the system will accept any valid FMGR command. The system expects use of the file manager program within a five minute period, and if this use does not occur, the FMGR program will automatically be terminated by RTE-II. When the FMGR program is terminated by RTE-II, the message in Figure D-2 is sent to the CRT, and the system is now ready to accept any RTE-II commands.

\- - -

```
SET TIME
:SV,4
TE,*****
TE,***** WELCOME TO THE SDS PLEASE TYPE RU,OPCOM WHEN YOU
TE,***** ARE READY TO BEGIN USING THE  SDS
TE,*****
::
: #END FMGR
```

Figure D-2. Automatic Time-Out of FMGR

The SDS can be run from either the FMGR program or RTE-II. If in FMGR mode, as implied by the prompt character colon, the user should type RU,OPCOM to run the SDS. If in RTE-II mode, as implied by no prompt character, the user should type *RU,OPCOM to run the SDS. Actually, any key could be struck in place of the asterisk but the asterisk is used in this description for the sake of simplicity. For a more detailed description of using the FMGR and RTE-II commands refer to Hewlett-Packard's manuals related to Real-Time Executive, Batch/Spool Monitor, and Operating System. Other manuals that may be helpful are the RTE-II and Batch-Spool Monitor Pocket Guide, and the Operating and Service Manual for the HP-2100 computer.

APPENDIX E

CREATING DISC FILE OPIN

APPENDIX E

CREATING DISC FILE OPIN

There are two methods, other than using the SDS, that may be used to create the disc file OPIN that can be used for input of source programs to the SDS. The first method is using the FMGR store command. This requires operating under the FMGR program, as designated by the colon prompt character, and that no disc file named OPIN exists. If a disc file named OPIN exists, it may be purged by the FMGR command PU,OPIN. The procedure in Figure E-1 can be used to create a disc file OPIN using the FMGR store command. Figure E-1 also exhibits the procedure for running this newly created file.

```
:PU,OPIN
:ST,1,OPIN
NEW$
ST1 AF 10 S TH 2$
ST2 AF 10 S$
END$
```

NOTE: After typing the END$ instruction the user must type a control D. This is accomplished by depressing the CTRL key and the character D simultaneously.

```
:RU,OPCOM
@
NEW$
INPUT FROM DISK??
YES
@
NEW$
@
ST1 AF 10 S TH 2$
@
ST2 AF 10 S$
@
END$
START EXP?
YES
END OF EXP.      1
*GO,OPCOM
:
```

The second method used to create the disc file OPIN is to use the RTE-II program EDITOR. This requires operating under the FMGR program, as designated by the colon prompt character, and that no disc file named OPIN exists. If a disc file named OPIN exists, it may be purged by the FMGR command PU, OPIN. The procedure shown in Figure E-2 can be used to create a disc file OPIN using the EDITOR program. Figure E-2 also exhibits the procedure for running this newly created file.

```
:PU,OPCOM
:RU,EDITOR
SOURCE FILE?
/                enter a blank and carriage return
EOF
/ NEW$           note that each line is preceded by a blank
/ ST1 AF 10 S TH  2$
/ ST 2 AF 10 S$
/ END$
/ECOPIN          ends editor creates file OPIN
END OF EDIT
:RU,OPCOM
@
NEW$
INPUT FROM DISC??
YES
@
NEW$
@
ST1 AF 10 S TH  2$
@
ST2 AF 10 S$
@
END$
START EXP?
YES
END OF EXP.      1
*GO,OPCOM
```

NOTE: Refer to Hewlett-Packard's Batch/Spool Monitor and Editor manuals for more information on store command and editor commands.

APPENDIX F

SDS ERRORS

APPENDIX F

SDS ERRORS

The operator communications program reports source language errors immediately following the line in which the error occurred. There are two basic error messages printed on the CRT when an error occurs during the source language input of an SDS program. These messages are, "Bad Operator", and "Outside Table". For information concerning any other error messages refer to the appropriate Hewlett-Packard manual.

The "Bad Operator" message occurs when the source language line violates the proper input format for instructions as described in Section V, the SDS Instruction Set. Some examples of these types of errors are shown in Figure F-1.

```
RU,OPCOM
@
NEW$
INPUT FROM DISK??
NO
@
ST1 AF TER 1 S TH 2$-----------embedded blank in AFTER
BAD OPERATOR
@
ST1 AFTER 1 S TH 2$
@
ST2 A 1 S TH  3$----------------AF instruction incomplete
BAD OPERATOR
@
ST2 AF 1 S TH 3$
@
ST3 AF 1 S TH4$-----------------no blank following TH
BAD OPERATOR
@
ST3 AF 1 S TH  4$
@
ST4 AF 1 S VAR A=1 TH 5$-------VAR instruction out of order
BAD OPERATOR
@
ST4 AF 1 S TH 5 VAR A=1$
@
ST5 CRT A AF 1 S$--------------AF instruction must follow ST5
BAD OPERATOR
@
ST5 AF 1 S CRT A$
@
END$
START EXP?
NO
*GO,OPCOM
:
```

Figure F-1 is not meant to give a complete list of all possible bad operator errors; it does, however, give examples of some of the most common errors. Note that when errors occur, retype the corrected line in order to correct the error. Review the SDS Instruction Set, Section V, if bad operator errors occur that are not easily recognized, checking for proper format of all instructions in the line in which the error occurred.

The "Outside Table" error occurs when a value has been assigned to a dimension statement, a state number, or a substate number that is larger than the maximum number allowed. Some examples of these types of errors are shown in Figure F-2.

```
:RU,OPCOM
@
NEW@
INPUT FROM DISC??
NO
@
ST31 AF 10 S TH 32$-------------state number larger than 30
OUTSIDE TABLE
@
ST1 AF 10 S TH 2  SS 31$-----------substate number larger than 30
OUTSIDE TABLE
@
ST1 AF 10 S TH 2 DIM A,300$------array size greater than 200 words
OUTSIDE TABLE
@
END$
START EXP?
NO
*GO, OPCOM
:
```

APPENDIX G

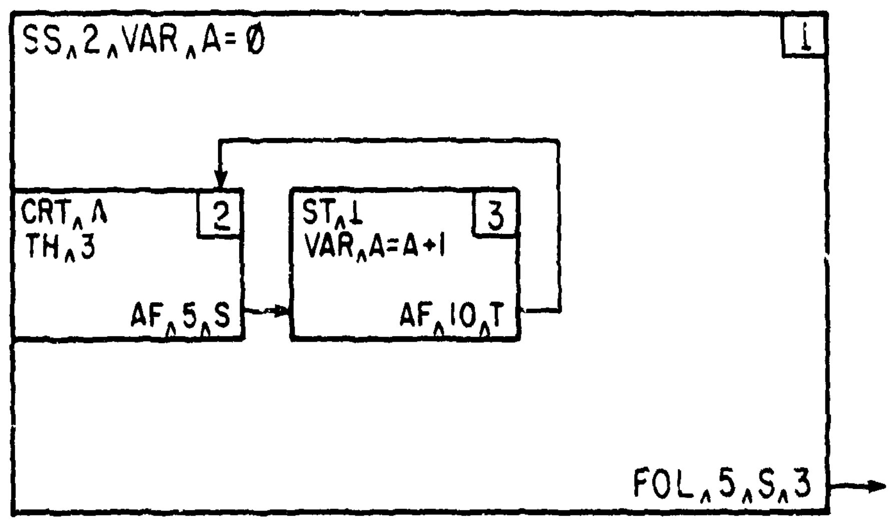
SAMPLE SDS PROGRAM RUN AND SAMPLE LOG

APPENDIX G

SAMPLE SDS PROGRAM RUN AND SAMPLE LOG

SAMPLE PROGRAM RUN

The program used in the sample is a fixed interval (FI) schedule with a master counter in state #1 to limit the number of reinforcements to five. The CRT instruction in state #3 is used to show that the program did execute states #2 and #3 five times. Figure G-1 describes the necessary state diagram and is also a copy of the console after the run is complete.




```
:RU, OPCOM
@
NEW$
INPUT FROM DISC??
YES
@
NEW$
@
ST1 FOL 5 S 3 SS 2 VAR A=0$
@
ST2 AF 5 S TH 3 CRT A$
@
ST3 AF 10 T TH 2 ST 1 VAR A=A+1$
@
END$
START EXP?
YES
   0
   1
   2            output from CRT A instruction
   3
   4
END OF EXP        1
*GO, OPCOM
:
```

Figure G-1. Diagram of Sample Program

SAMPLE LOG

The sample log shown in Figure G-2 was written on the magnetic tape during the running of the program described in Figure G-1. Table G-1 describes the log events sequentially in terms of log entries. Refer to Section VII, the SDS Log, for contents of various log entries.

Table G-I

Log Events from Log in Figure G-2

Log entry #	Event	Elapsed time since previous entry
1	start of experiment	----
2	start of state #1	10 ms
3	start of state #2	10 ms
4	end of state #2	5 seconds
5	start of state #3	10 ms
6	end of state #3	110 ms
7	start of state #2	10 ms
8	end of state #2	5 seconds
9	start of state #3	10 ms
10	end of state #3	110 ms
11	start of state #2	10 ms
12	end of state #2	5 seconds
13	start of state #3	10 ms
14	end of state #3	110 ms
15	start of state #2	10 ms
16	end of state #2	5 seconds
17	start of state #3	10 ms
18	end of state #3	110 ms
19	start of state #3	10 ms
20	end of state #2	5 seconds
21	start of state #3	10 ms
22	end of state #1	25 seconds 640 ms since it began
23	end of state #3	10 ms
24	end of experiment	25 seconds 660 ms since it began

SAMPLE

```
REC# 00001
001000 000001 000000 000000 000126 000010 000007 000010*
000335 002000 000001 000001 000000 000127 000010 000007*
000010 000335 002000 000001 000002 000000 000130 000010*
000007 000010 000335 003000 000001 000002 000030 000130*
000015 000007 000010 000335 002000 000001 000003 000000*
000131 000015 000007 000010 000335 003000 000001 000003*
000000 000000 000016 000007 000010 000335 002000 000001*
000002 000000 000001 000016 000007 000010 000335 003000*
000001 000002 000000 000001 000023 000007 000010 000335*
002000 000001 000003 000000 000002 000023 000007 000010*
000335 003000 000001 000003 000000 000015 000023 000007*
000010 000335                                          *
REC# 00002
002000 000001 000002 000000 000016 000023 000007 000010*
000335 003000 000001 000002 000000 000016 000030 000007*
000010 000335 002000 000001 000003 000000 000017 000030*
000007 000010 000335 003000 000001 000003 000000 000032*
000030 000007 000010 000335 002000 000001 000002 000000*
000033 000030 000007 000010 000335 003000 000001 000002*
000000 000033 000035 000007 000010 000335 002000 000001*
000003 000000 000034 000035 000007 000010 000335 003000*
000001 000003 000000 000047 000035 000007 000010 000335*
002000 000001 000002 000000 000050 000035 000007 000010*
000335 003000 000001 000002 000000 000050 000042 000007*
000010 000335                                          *
REC# 00003
002000 000001 000003 000000 000051 000042 000007 000010*
000335 003000 000001 000001 000000 000063 000042 000007*
000010 000335 003000 000001 000003 000000 000064 000042*
000007 000010 000335 005000 000001 000000 000000 000064*
000042 000007 000010 000335                            *
```

Figure G-2. Octal Listing of SDS Log

The State Diagram System (SDS) was developed to solve this problem. SDS is a tool that can be used by investigators in designing and running psychophysical experiments on Hewlett-Packard's HP-2100 series computers. SDS, as presently designed, is capable of running only those experiments that use discrete inputs and outputs. The system offers the investigator a high level language with which he is already familiar or can easily learn, thus removing the burden of solving these types of problems using more complex computer languages. Written in FORTRAN IV language SDS is an interactive system that does not require assembling or compiling of its programs. The system accepts source language statements from either the system console or disc files and allows the program to be run immediately upon completion of this input process. While SDS does not solve all of the problems encountered in computerizing psychological experiments, its modular design should ease such future modifications as dealing with continuous variables, calling external programs, and controlling multiple experiments.

The State Diagram System (SDS) was developed to solve this problem. SDS is a tool that can be used by investigators in designing and running psychophysical experiments on Hewlett-Packard's HP-2100 series computers. SDS, as presently designed, is capable of running only those experiments that use discrete inputs and outputs. The system offers the investigator a high level language with which he is already familiar or can easily learn, thus removing the burden of solving these types of problems using more complex computer languages. Written in FORTRAN IV language SDS is an interactive system that does not require assembling or compiling of its programs. The system accepts source language statements from either the system console or disc files and allows the program to be run immediately upon completion of this input process. While SDS does not solve all of the problems encountered in computerizing psychological experiments, its modular design should ease such future modifications as dealing with continuous variables, calling external programs, and controlling multiple experiments.